buzdağı

Yayın Numarası: 1902
1. Baskı: Eylül 2019
ISBN 978-605-69024-2-0

Sertifika Numarası: 41825
Buzdağı Yayınevi
Altay Mahallesi 2668. Cadde
Fırat Life Style Rezidans 1F/61
Eryaman/ANKARA
t: +90 312 219 77 98
f: +90 312 219 55 43
info@buzdagiyayinevi.com

Baskı
Ankamat Matbaacılık
13256

Original English language edition published by Manning Publications
Copyright © 2017 by Manning Publications.
© 2019 Türkçe yayın hakları Buzdağı Yayınevi'ne aittir.
Bu kitabın hiçbir bölümü, yazarın ve yayınevinin izni alınmadan
basılı ve dijital olarak çoğaltılamaz, yayınlanamaz.

Python ile
Derin Öğrenme

François Chollet

Genel Yayın Yönetmeni
Fatih ÖZDEMİR

Yazar
François CHOLLET

Çevirmen
Bilgin AKSOY

Editör
Birol KUYUMCU

Son Okuma
Kaan Can YILMAZ

Dizgi
Veysel TOPRAK

İçindekiler

Ön Söz ix

Çevirmen Ön Sözü xi

Teşekkür xiii

Bu Kitap Hakkında xv

Yazar Hakkında xix

Kapak Hakkında xx

I Derin Öğrenmenin Temelleri 1

1 Derin Öğrenme Nedir? 3

1.1 Yapay Zekâ, Makine Öğrenmesi ve Derin Öğrenme 4
1.1.1 Yapay Zekâ 4
1.1.2 Makine Öğrenmesi 5
1.1.3 Veriden Gösterimi Öğrenmek 6
1.1.4 Derinlemesine Derin Öğrenme 8
1.1.5 Derin Öğrenmeyi Üç Şekille Anlamak 9
1.1.6 Derin Öğrenme Bugüne Kadar Neler Başardı? 12
1.1.7 Kısa Vadede Olacaklar ile İlgili Yapılan Abartılı Açıklamalara İnanmayın! 12
1.1.8 Yapay Zekânın Vaatleri 13
1.2 Makine Öğrenmesi Tarihine Kısa Bir Bakış 14
1.2.1 Olasılıksal Modelleme 15
1.2.2 İlk Sinir Ağları 15
1.2.3 Kernel Metotları 16

1.2.4 Karar Ağaçları, Rastgele Orman ve Gradyan Hızlandırma Makineleri . . . 17
1.2.5 Sinir Ağlarına Dönüş . . . 18
1.2.6 Derin Öğrenmeyi Farklı Kılan Şey Nedir? . . . 19
1.2.7 Çağdaş Makine Öğrenmesi Bakışı . . . 20
1.3 Neden ve Neden Şimdi Derin Öğrenme? . . . 21
1.3.1 Donanım . . . 21
1.3.2 Veri . . . 22
1.3.3 Algoritmalar . . . 23
1.3.4 Yeni Yatırım Dalgası . . . 24
1.3.5 Derin Öğrenmenin Yaygınlaşması . . . 24
1.3.6 Sona Erecek mi? . . . 25

2 Derin Öğrenmenin Matematiksel Temelleri 27
2.1 Sinir Ağlarına İlk Bakış . . . 28
2.2 Sinir Ağları için Veri Gösterimleri . . . 32
2.2.1 Skalerler (0B Tensörler) . . . 32
2.2.2 Vektörler (1B Tensörler) . . . 32
2.2.3 Matrisler (2B Tensörler) . . . 33
2.2.4 3B ve Daha Fazla Boyutlu Tensörler . . . 33
2.2.5 Anahtar Özellikler . . . 34
2.2.6 Numpy ile Tensör İşlemleri . . . 35
2.2.7 Veri Yığını Notasyonu . . . 36
2.2.8 Veri Tensörleri için Gerçek Dünya Örnekleri . . . 37
2.2.9 Vektör Verisi . . . 37
2.2.10 Zaman Serisi Verisi ya da Ardışık Veri . . . 38
2.2.11 Görüntü Verisi . . . 38
2.2.12 Video Verisi . . . 39
2.3 Sinir Ağları Alet Çantası: Tensör İşlemleri . . . 40
2.3.1 Eleman Bazlı İşlemler . . . 40
2.3.2 Yayma Operasyonu . . . 41
2.3.3 Tensör İç Çarpımı . . . 42
2.3.4 Tensör Şekil Değiştirme . . . 45
2.3.5 Tensör İşlemlerinin Geometrik Gösterimi . . . 46
2.3.6 Derin Öğrenmenin Geometrik Gösterimi . . . 47
2.4 Sinir Ağlarının Motoru (Gradyan Tabanlı Eniyileme) . . . 48
2.4.1 Türev Nedir? . . . 49
2.4.2 Tensör İşleminin Türevi: Gradyan . . . 50
2.4.3 Stokastik Gradyan İnişi . . . 51
2.4.4 Zincir Türev: Geriye Yayılım Algoritması . . . 54
2.5 İlk Örneğe Yeniden Bakış: . . . 55

3 Sinir Ağlarına Giriş 57
- 3.1 Sinir Ağlarının Yapısı 58
  - 3.1.1 Katmanlar: Derin Öğrenmenin Yapı Taşları 58
  - 3.1.2 Modeller: Katman Ağı 60
  - 3.1.3 Kayıp Fonksiyonları ve Eniyileme Algoritmaları: Eğitim Sürecinin Yönetilmesinin Anahtarları 60
- 3.2 Keras'a Giriş 61
  - 3.2.1 Keras, TensorFlow, Theano ve CNTK 62
  - 3.2.2 Keras ile Geliştirme: Hızlı Başlangıç 63
- 3.3 Derin Öğrenme Ortamının Kurulması 65
  - 3.3.1 Jupyter Not Defterleri: Derin Öğrenme Denemeleri Yapmanın En Çok Tercih Edilen Yolu 65
  - 3.3.2 Keras'ı Çalıştırmak: İki Seçenek 66
  - 3.3.3 Derin Öğrenmeyi Bulut Üzerinde Kullanmak: Artıları ve Eksileri 66
  - 3.3.4 Derin Öğrenme için En İyi GPU nedir? 67
- 3.4 Film Kritiklerini Sınıflandırma: İkili Sınıflandırma Örneği 67
  - 3.4.1 IMDB Veri Seti 67
  - 3.4.2 Verileri Hazırlamak 69
  - 3.4.3 Ağınızı İnşa Etmek 70
  - 3.4.4 Yaklaşımınızı Doğrulamak 74
  - 3.4.5 Öneğitimli Ağı Yeni Veriler Üzerinde Tahmin için Kullanmak 77
  - 3.4.6 İleri Gözlemler 78
  - 3.4.7 Özet 78
- 3.5 Haber Sınıflandırma: Çoklu Sınıflandırma Örneği 79
  - 3.5.1 Reuters Veri Seti 79
  - 3.5.2 Verileri Hazırlamak 80
  - 3.5.3 Ağı İnşa Etmek 81
  - 3.5.4 Yaklaşımınızı Doğrulamak 82
  - 3.5.5 Yeni Verilerde Tahmin Oluşturma 85
  - 3.5.6 Etiketleri ve Kaybı İşlemenin Alternatif Yolu 86
  - 3.5.7 Yeterince Büyük Ara Katmanlara Sahip Olmanın Önemi . 86
  - 3.5.8 İleri Gözlemler 87
  - 3.5.9 Özet 87
- 3.6 Ev Fiyatı Tahmini: Doğrusal Bağlanım Örneği 87
  - 3.6.1 Boston Gayrimenkul Fiyatları Veri Seti 88
  - 3.6.2 Verilerin Hazırlanması 89
  - 3.6.3 Ağı İnşa Etmek 89
  - 3.6.4 K-fold Doğrulama ile Yaklaşımımızı Doğrulamak 90
  - 3.6.5 Özet 95

4 Makine Öğrenmesinin Temelleri 97
4.1 Makine Öğrenmesinin Dört Kategorisi 98
4.1.1 Denetimli Öğrenme 98
4.1.2 Denetimsiz Öğrenme 98
4.1.3 Yarı-Denetimli Öğrenme 99
4.1.4 Pekiştirmeli Öğrenme 99
4.2 Makine Öğrenme Modellerinin Değerlendirilmesi 101
4.2.1 Eğitim, Doğrulama ve Test Veri Setleri 101
4.2.2 Aklınızdan Çıkarmamanız Gerekenler 105
4.3 Veri Önişleme, Öznitelik Çıkarımı ve Öznitelik Öğrenme 106
4.3.1 Sinir Ağları için Veri Önişleme 106
4.3.2 Öznitelik Çıkarımı 107
4.4 Aşırı Uydurma ve Yetersiz Uydurma 109
4.4.1 Ağı Küçültmek 110
4.4.2 Ağırlıkların Düzenlileştirilmesi 112
4.4.3 İletim Sönümü 114
4.5 Makine Öğrenmesi İş Akışı 117
4.5.1 Problem Tanımı ve Veri Seti Oluşturma 117
4.5.2 Başarı Ölçüsünü Seçmek 118
4.5.3 Değerlendirme Protokolü Seçimi 118
4.5.4 Verileri Hazırlama 118
4.5.5 Ağınızı Geliştirmek 119
4.5.6 Aşırı Uyduran Bir Model Geliştirmek 120
4.5.7 Düzenlileştirme ve Hiperparametreleri Ayarlama 121

II Uygulamalı Derin Öğrenme 123

5 Bilgisayarlı Görü için Derin Öğrenme 125
5.1 Evrişimli Sinir Ağlarına Giriş 126
5.1.1 Evrişim İşlemi 129
5.1.2 En Büyükleri Biriktirme 134
5.2 Küçük Bir Veri Setinde Evrişimli Sinir Ağını En Baştan Eğitmek 136
5.2.1 Küçük Veri Setleri için Derin Öğrenmenin Uygunluğu . . 136
5.2.2 Veri Setini İndirmek 137
5.2.3 Ağı İnşa Etmek 140
5.2.4 Veri Önişleme 142
5.2.5 Veri Seti Çeşitlendirme 147
5.3 Öneğitimli Evrişimli Sinir Ağı Kullanmak 151
5.3.1 Öznitelik Çıkarımı 152
5.3.2 Hassas Ayar 163

5.3.3 Özet 170
5.4 Evrişimli Sinir Ağının Öğrendiklerini Görselleştirmek 170
5.4.1 Ara Çıktıları Görselleştirmek 171
5.4.2 Evrişim Filtrelerini Görselleştirmek 177
5.4.3 Sınıf Aktivasyon Isı Haritalarını Görselleştirmek 183

6 Metin ve Diziler için Derin Öğrenme 189
6.1 Metin Verisi ile Çalışmak 190
6.1.1 Bir-Elemanı-Bir Kodlanmış Kelime ve Karakterler 192
6.1.2 Kelime Gömülmelerini Kullanma 194
6.1.3 Ham Veriden Kelime Gömülmelerine 199
6.1.4 Özet 207
6.2 Yinelemeli Sinir Ağlarını Anlamak 208
6.2.1 Keras'ta Yinelemeli Katman 211
6.2.2 LSTM ve GRU Katmanlarını Anlamak 215
6.2.3 Keras LSTM Örneği 218
6.2.4 Özet 220
6.3 Yinelemeli Sinir Ağlarının İleri Seviye Kullanımı 220
6.3.1 Hava Sıcaklığı Tahmin Problemi 220
6.3.2 Veriyi Hazırlamak 223
6.3.3 Akıl Yürütme, Makine Öğrenmesi Olmaksızın Temel Bir Model 226
6.3.4 Temel Makine Öğrenmesi Yaklaşımı 227
6.3.5 İlk Yinelemeli Temel Model 229
6.3.6 Yinelemeli İletim Sönümü ile Aşırı Uydurmayla Mücadele 231
6.3.7 Üst Üste Yinelemeli Katman Kullanımı 232
6.3.8 İki Yönlü RNN'ler 234
6.3.9 İleriye Gitmek 238
6.3.10 Özet 238
6.4 Evrişimli Ağlarla Dizi İşleme 239
6.4.1 Dizi Verilerde 1B Evrişim İşlemini Anlamak 240
6.4.2 Dizi Verilerde 1B Biriktirme 240
6.4.3 1B Evrişimli Sinir Ağını Gerçekleştirmek 241
6.4.4 CNN ve RNN'leri Birleştirerek Uzun Zaman Serilerini İşlemek 243
6.4.5 Özet 247

7 İleri Derin Öğrenme, En İyi Uygulamalar 249
7.1 Sequential Modelin Ötesine Gitmek: Keras functional API 249
7.1.1 functional API'ye Giriş 252
7.1.2 Çoklu Girdili Modeller 254

7.1.3 Çoklu Çıktılı Modeller . . . 257
7.1.4 Yönlü Çevrimsiz Çizgesel Katmanlar . . . 259
7.1.5 Ağırlıkları Paylaşmak . . . 263
7.1.6 Katman Olarak Model Kullanmak . . . 264
7.1.7 Özet . . . 265
7.2 Keras Geri Çağırmaları ve TensorBoard Kullanarak Derin Öğrenme Modellerini İncelemek ve Gözlemek . . . 266
7.2.1 Model Eğitilirken Geri Çağırmalar . . . 266
7.2.2 TensorBoard'a Giriş - TensorFlow'un Görselleştirme Kütüphanesi . . . 270
7.2.3 Özet . . . 276
7.3 Modellerin Suyunu Çıkarmak . . . 277
7.3.1 İleri Mimari Desenler . . . 277
7.3.2 Hiperparametre Eniyilemesi . . . 280
7.3.3 Topluluk Yöntemleri . . . 282
7.3.4 Özet . . . 284

8 Üretici Derin Öğrenme 287
8.1 LSTM Kullanarak Metin Üretme . . . 289
8.1.1 Üretici Yinelemeli Ağların Tarihi . . . 289
8.1.2 Dizi Verisi Nasıl Üretilir? . . . 290
8.1.3 Örnekleme Stratejisinin Önemi . . . 290
8.1.4 LSTM Kullanarak Karakter Seviyesinde Metin Üretme . . . 292
8.1.5 Özet . . . 298
8.2 DeepDream . . . 298
8.2.1 Keras'ta DeepDream Gerçekleştirmesi . . . 299
8.2.2 Özet . . . 305
8.3 Sinirsel Stil Aktarımı . . . 306
8.3.1 `content` Kaybı . . . 307
8.3.2 `style` Kaybı . . . 307
8.3.3 Keras'ta Sinirsel Stil Aktarımı . . . 308
8.3.4 Özet . . . 314
8.4 Değişimsel Otokodlayıcılarla Resim Üretme . . . 316
8.4.1 Resim Saklı Uzayından Örnekleme . . . 316
8.4.2 Kavram Vektörüyle Resim Düzenleme . . . 317
8.4.3 Değişimsel Otokodlayıcılar . . . 318
8.4.4 Özet . . . 325
8.5 Çekişmeli Üretici Ağlara Giriş . . . 326
8.5.1 GAN'ın Şematik Gösterimi . . . 327
8.5.2 İpuçları . . . 328
8.5.3 Üretici . . . 329

8.5.4 Ayırıcı 330
8.5.5 Çekişmeli Ağ 331
8.5.6 DCGAN'ı Eğitmek 332
8.5.7 Özet 334

9 Sonuç 337

9.1 Kilit Kavramların Tekrarı 338
9.1.1 YZ'ya Farklı Yaklaşımlar 338
9.1.2 Makine Öğrenmesi Alanında Derin Öğrenmeyi Özel Yapan Şey Nedir? 338
9.1.3 Derin Öğrenme Hakkında Nasıl Düşünülmeli? 339
9.1.4 Kilit Teknolojiler 340
9.1.5 Makine Öğrenmesi İş Akışı 341
9.1.6 Kilit Ağ Mimarileri 342
9.1.7 Olasılık Uzayı 346
9.2 Derin Öğrenmenin Sınırları 348
9.2.1 Makine Öğrenmesi Modellerini İnsanlaştırma Riski 349
9.2.2 Sınırlı Genelleştirme ve Tamamen Genelleştirme 351
9.2.3 Özet 352
9.3 Derin Öğrenmenin Geleceği 352
9.3.1 Program Olarak Modeller 353
9.3.2 Geriye Yayılım ve Türevlenebilir Katmanların Ötesi . . . 355
9.3.3 Otomatik Makine Öğrenmesi 355
9.3.4 Yaşamboyu Öğrenme ve Altprogramların Yeniden Kullanımı356
9.3.5 Uzun Dönemli Öngörüler 357
9.4 Bilgilerinizin Güncel Kalmasını Sağlamak 359
9.4.1 Kaggle Kullanarak Gerçek Dünya Sorunlarında Pratik Yapın 359
9.4.2 arXiv Üzerinden Son Gelişmeleri Takip Edin 359
9.4.3 Keras Ekosistemini İnceleyin 360
9.5 Son Sözler 360

A Ubuntu'da Keras ve Gerekli Kütüphaneleri Kurmak 361

A.1 Python Bilimsel Kütüphanelerini Kurmak 362
A.2 GPU Desteğini Kurmak 363
A.3 Theano'yu Kurmak (İsteğe Bağlı) 364
A.4 Keras'ı Kurmak 365

B AWS EC2'de Jupyter Not Defteri Kullanmak 367

B.1 Jupyter Not Defteri Nedir? Neden AWS GPU'larında Jupyter Kullanılmalıdır? 367

B.2 Derin Öğrenme için AWS'de Jupyter'i Neden Kullanmamalısınız? 368
B.3 AWS GPU Örneğini Kurmak 368
B.4 Jupyter'i Yapılandırmak 370
B.5 Keras'ı Kurmak 372
B.6 Port Yönlendirmek 373
B.7 Kendi Bilgisayarınızdaki Tarayıcıdan Jupyter'i Kullanmak 373

Ön Söz

Eğer bu kitabı elinizde bulunduruyorsanız muhtemelen derin öğrenmenin yakın geçmişte yapay zekâ alanındaki sıra dışı ilerlemesi hakkında bilgi sahibisinizdir. Sadece 5-6 yılda neredeyse kullanışsız görüntü ve konuşma tanımadan bu görevlerde insanüstü başarılar elde ettik.

Bu ani gelişmenin etkileri neredeyse tüm endüstriyi etkiledi. Ama derin öğrenme teknolojilerini, çözülebilecek tüm problemlerde kullanabilmek için araştırmacı ve lisansüstü öğrencisi olmayanlarda dahil olabildiğince fazla insan tarafından ulaşılabilir yapmalıyız. Tüm potansiyeline ulaşana kadar herkes tarafından ulaşılabilir yapmalıyız.

Mart 2015'te Keras derin öğrenme kütüphanesinin ilk versiyonunu yayımladığımda YZ'yı herkese ulaştırmak aklımda yoktu. Yıllardır makine öğrenmesi alanında araştırma yapıyordum ve Keras'ı kendi denemelerimde yardımcı olması için geliştirmiştim. 2015'ten 2016'ya on binlerce yeni insan derin öğrenme alanına girdi -hâlâ da girmeye devam ediyor- ve çoğu başlangıç için en kolay kütüphane olduğundan Keras'ı seçtiler. Beklenmedik Keras kullanımı beni YZ'nın ulaşılabilirliği hakkında düşünmeye sevk etti. Bu teknolojileri yaymanın daha kullanışlı ve değerli olduğunu anlamama yol açtı. Ulaşılabilirlik Keras geliştirme sürecinin ana hedefi oldu ve birkaç yılda Keras geliştiricileri çok önemli başarılara imza attı. Böylece on binlerce insan daha önce var olduğunu bile bilmediğimiz sorunları çözdü.

Elinizdeki bu kitap, olabildiğince çok insana derin öğrenmeyi ulaşılabilir kılmanın başka bir adımıdır. Keras, her zaman derin öğrenmenin temellerini, Keras kullanım şeklini ve en iyi derin öğrenme uygulamalarını aynı anda aktaracak bir kaynağa ihtiyaç duyuyordu. Bu kitap, böyle bir kaynağı oluşturmak için var gücümle çalışmamın eseridir. Bu kitabı derin öğrenmenin arkasındaki kavramlara, gerçeklenmesine ve mümkün olduğunca ulaşılabilir olmasına odaklanarak yazdım. Bunu yaparken hiçbir şeyi basitleştirmedim çünkü derin öğrenmenin arkasındaki fikrin çok basit olduğunu düşünüyorum. Bu kitabı değerli bulmanızı ve sizin için sorun teşkil eden problemlerin çözümünde akıllı uygulamalar geliştirmenizi sağlamasını umuyorum.

Bu Kitap Hakkında

Bu kitap, derin öğrenmeyi sıfırdan öğrenecek ya da bilgilerini artıracak herkes için yazılmıştır. Makine öğrenmesi mühendisi, yazılım geliştirici veya üniversite öğrencisi de olsanız bu kitapta değerli sayfalar bulacaksınız.

Bu kitap, uygulamalarla ve kod yazarak derin öğrenmeyi öğretmeye çalışmaktadır. Matematiksel notasyon yerine sayısal kavramları kod örnekleriyle açıklayarak makine öğrenmesi, derin öğrenmenin temel fikirleri hakkında uygulamalı bilinç oluşturmaya çalışmaktadır.

Ayrıntılı olarak açıklamaların eklendiği 30'dan fazla kod örneğinden, pratik tavsiyelerden ve derin öğrenmeyi somut problemlerin çözümünde kullanmaya başlamak için gerekli her şeyin detaylı açıklamalarından öğreneceksiniz.

Kod örnekleri Python derin öğrenme kütüphanesi Keras'ı arka planda TensorFlow'la kullanmaktadır. Keras en popüler ve hızlı gelişen derin öğrenme kütüphanelerinden biridir ve genelde yeni başlayanlara yaygın olarak tavsiye edilmektedir.

Bu kitabı okuduktan sonra derin öğrenmenin ne olduğu ne zaman uygulanabilir olduğu ve kısıtları hakkında sağlam bir temele sahip olacaksınız. Makine öğrenmesi problemlerine yaklaşmada standart bir yol haritasını ve sıkça karşılaşılan sorunlara nasıl çözüm getireceğinizi öğreneceksiniz. Görüntü sınıflandırma, zaman serisi tahmini, duygu analizi, resim ve metin üretme gibi bilgisayarlı görüden doğal dil işlemeye kadar birçok gerçek hayat probleminde Keras'ı kullanabileceksiniz.

Bu Kitabı Kimler Okumalı?

Bu kitap, Python programlama tecrübesi olanlardan makine öğrenmesi ve derin öğrenmeye başlamak isteyenler için yazılmıştır. Ama bu kitap diğer birçok okuyucuya da hitap etmektedir:

- Makine öğrenmesine aşina bir veri bilimciyseniz bu kitap size makine öğrenmesinin en hızlı genişleyen ve en önemli alt alanı derin öğrenme hakkında sağlam ve uygulamalı bir giriş imkânı sağlayacaktır.

- Derin öğrenme uzmanıysanız ve Keras kütüphanesine başlangıç arıyorsanız bu kitap Keras hakkındaki en iyi kaynak olacaktır.
- Lisansüstü öğrencisiyseniz ve derin öğrenme çalışıyorsanız bu kitapta eğitiminizi pratik uygulamalarla destekleyecek, derin sinir ağlarının davranışları hakkında bilinçlenecek ve en iyi uygulamalar hakkında bilgi sahibi olacaksınız.

Normalde, işi geliştirme yapmak olmayan diğer teknik insanlarda bu kitaptaki temel ve ileri seviyeli derin öğrenme konseptlerini faydalı bulacaktır.

Keras kullanabilmek için Python'da yeterince iyi olmalısınız. Her ne kadar gerekli olmasa da Numpy kütüphanesine aşina olmanız iyi olacaktır. Makine öğrenmesi ve derin öğrenmeyle ilgili daha önce tecrübe sahibi olmanıza gerek yok: Bu kitap sıfırdan tüm temelleri kapsamaktadır. İleri seviye matematik temeline ihtiyacınız yok lise matematik bilgisi yeterli olacaktır.

Yol Haritası

Bu kitap iki kısma ayrılmıştır. Daha önce makine öğrenmesi tecrübeniz yoksa birinci kısmı okumanızı şiddetle tavsiye ederim. Basit örneklerle başlayacağız ve ilerledikçe en iyi sonuçları elde eden tekniklere bakacağız.

Kısım-I makine öğrenmesi ve sinir ağlarına başlamak için gerekli tüm kavramları ve tanımları barındıran derin öğrenmeye girişi içermektedir:

- Bölüm-1 YZ, makine öğrenmesi ve derin öğrenmenin temellerini ve gerekli içerikleri sunmaktadır.
- Bölüm-2 tensör, tensör işlemleri, gradyan inişi, geriye yayılım gibi derin öğrenmenin temel kavramlarını açıklamaktadır. Bu bölüm ayrıca çalışan bir sinir ağının ilk örneğini içermektedir.
- Bölüm-3 sinir ağlarına başlamak için geriye kalan her şeyi kapsamaktadır: Keras'a giriş, derin öğrenme kütüphanesi seçimi, çalışma ortamının kurulması ve detaylı açıklanan temel üç kod örneğini açıklamaktadır. Bu bölümün sonunda, sınıflandırma ve bağlanım görevlerinde basit bir sinir ağını eğitebilecek ve ağınızı eğitirken arka planda neler olduğu hakkında sağlam bir fikre sahip olacaksınız.
- Bölüm-4 standart makine öğrenmesi iş akışını incelemektedir. Aynı zamanda gizli tuzakları ve bunların çözüm yollarını öğreneceksiniz.

Kısım-II bilgisayarlı görü ve doğal dil işlemede derin öğrenme uygulamalarını derinlemesine açıklamaktadır. Bu kısımda açıklanan örnekler derin öğrenmede gerçek hayatta karşılaşacağınız sorunların çözümünde şablon olacaktır:

- Bölüm-5 görüntü sınıflandırmaya odaklanarak uygulamalı bilgisayarlı görü örneklerini açıklamaktadır.
- Bölüm-6 metin ve zaman serisi gibi dizi verilerini işlemede kullanacağınız teknikleri anlatmaktadır.
- Bölüm-7 en iyi sonuçları elde eden derin öğrenme modellerini açıklamaktadır.
- Bölüm-8 görüntü ve resim üreten ve bazen oldukça ilginç sonuçlar elde edebilen üretici modelleri kapsamaktadır.
- Bölüm-9 bu kitapta neler öğrendiğinize, derin öğrenmenin kısıtlarına ve gelecekte olabileceklere ayrılmıştır.

Yazılım/Donanım Gereksinimleri

Bu kitaptaki tüm kod örnekleri açık kaynaklı ve ücretsiz indirebileceğiniz Keras derin öğrenme kütüphanesi kullanılarak hazırlanmıştır (https://keras.io). UNIX işletim sistemine sahip bir bilgisayara ihtiyacınız olacak, Windows işletim sistemiyle kullanmak mümkün olsa da tavsiye etmiyorum. Ek-A'da kurulum için gerekli adımları bulabilirsiniz.

Makinenizde bir NVIDIA GPU bulunmasını tavsiye ediyorum, mesela TITAN X. Bu bir zorunluluk değil ama kod örneklerini çok daha hızlı çalıştırmanızı ve daha iyi tecrübe etmenizi sağlar. Bölüm-3.3'te derin öğrenme iş istasyonunu kurmak için daha ayrıntılı bilgiler bulabilirsiniz.

Eğer NVIDIA GPU'ya sahip bir iş istasyonuna sahip değilseniz bulut üzerinde çalışmayı tercih edebilirsiniz. Google Cloud (NVIDIA Tesla K 80 bulunduran n1-standart-8 makineler gibi) veya Amazon Web Services (p2.xlarge makineler gibi) kullanabilirsiniz. Ek-B'de AWS üzerinde Jupyter not defteri kullanarak tarayıcınızda nasıl çalışabileceğinizi bulacaksınız.

Kaynak Kodlar

Bu kitaptaki tüm kod örneklerine http://www.manning.com/books/deep-learning-with-python bağlantısındaki kitabın web sayfasından veya GitHub (https://github.com/fchollet/deep-learning-with-python-notebooks) üzerinden ulaşabilirsiniz.

Kitap Forumu

Bu kitabı satın alarak Manning yayıncılık tarafından sağlanan kitap hakkında yorum yapabileceğiniz, teknik sorular sorabileceğiniz, yazar ve diğer kullanıcılardan yardım alabileceğiniz özel bir foruma ücretsiz ulaşma hakkına sahip oldunuz. https://forums.manning.com/forums/deep-learning-with-python bağlantısından foruma ulaşabilirsiniz. Manning forum kurallarına ve içeriğin kullanımındaki etik kurallara https://forums.manning.com/forums/about bağlantısından ulaşabilirsiniz.

Manning okuyucularımıza, diğer okuyucular ve yazarla anlamlı diyaloglar kurulabileceği bir ortam sağlayacağını taahhüt eder. Bu taahhüt, yazarın isteğine bağlı (ücretsiz olduğundan) olduğundan tamamen taahhüt edilemez. Zorlu sorular sorarak yazarın dikkatini celbetmenizi öneriyoruz. Kitap yayında olduğu sürece forum ve önceki tartışmalar yayıncının web sayfasında ulaşılabilir olacaktır.

Yazar Hakkında

François Chollet Google Mountain View,CA'da derin öğrenme üzerine çalışmaktadır. Keras derin öğrenme kütüphanesinin oluşturucusu ve aynı zamanda TensorFlow makine öğrenmesi kütüphanesinin de geliştirilmesine katkı vermiştir. Aynı zamanda bilgisayarlı görü ve nedensellikte makine öğrenilmesinin kullanılması başta olmak üzere derin öğrenme araştırmaları yapmaktadır. Makaleleri başta Conference on Computer Vision and Pattern Recognition (CVPR), Conference and Workshop on Neural Information Processing Systems (NIPS) ve International Conference on Learning Representations (ICLR) olmak üzere alanın önde gelen konferanslarında yayınlanmıştır.

Bölüm 1

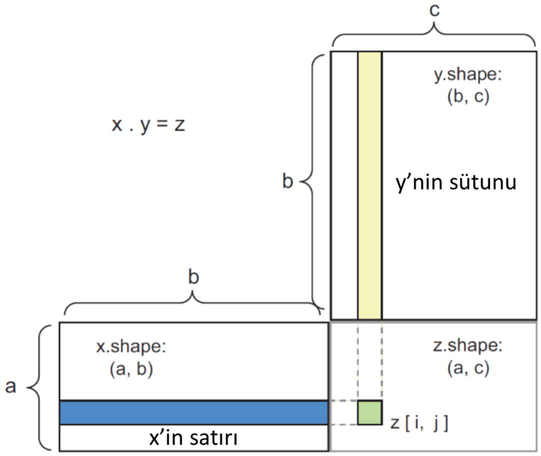
Derin Öğrenme Nedir?

Bölüm Kapsamı:

- **Temel kavramların en basit tanımları,**
- **Makine öğrenmesinin tarihçesi,**
- **Derin öğrenmeyi bu kadar popüler yapan ana etkenler ve geleceği hakkındaki beklentiler.**

Geçtiğimiz son birkaç yıl içerisinde yapay zekâ herkesin dikkatini çeken ve ilgi odağı olan bir konu hâline geldi. Makine öğrenmesi, derin öğrenme ve yapay zekâ ile ilgili sayısız makale yayımlanmıştır ki bunlardan bazıları teknoloji odaklı bile olmayan mecralarda gerçekleşmiştir. Gelecekte akıllı sohbet robotlarının, otonom araçların ve sanal asistanların yaygın kullanılacağını hatta biraz hayali gelse de insanlar tarafından yapılan rutin işlerin azalacağı ve birçok ekonomik işin robotlar tarafından yerine getirilmesini bekliyoruz. Makine öğrenmesi ile uğraşanların bugün veya gelecekte dünyayı değiştiren ilerlemeler kaydedebilmesi için karmaşa içinde örüntüyü tanımlayabilmesi önemlidir. Geleceğimiz aslında az da olsa buna bağlı ki bu kitabı okuduktan sonra gelecekte bu dönüşümü sağlayan yapay zekâ geliştiricilerinden birisi olabilirsiniz! O zaman şimdi derin öğrenmenin bugüne kadar neler başardığını, başardıklarının önemini, nereye doğru gittiğini ve vadedilenlere inanmanın doğru olup olmadığını cevaplayalım.

Bu bölüm yapay zekâ, makine öğrenmesi ve derin öğrenmenin temel içeriklerini aktarmaktadır.



Şekil 2.5: Matris iç çarpım diyagramı

2B tensörler için ifade ettiğimiz bu boyut uyumunu daha büyük boyutlu tensörlere de genelleştirmek mümkündür:

```
(a, b, c, d) . (d,) -> (a, b, c)
(a, b, c, d) . (d, e) -> (a, b, c, e)
```

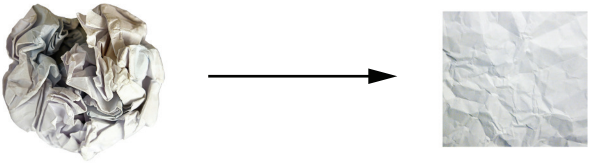
2.3.4 Tensör Şekil Değiştirme

Üçüncü olarak anlamanız gereken tensör işlemi *tensör şekil değiştirme* işlemidir. İlk yapay sinir ağı örneğimizde Dense katmanında kullanmamış olsak da ilk ağımıza girdi olarak göndermeden önce rakamları önişlemden geçirirken kullandık.

```
train_images = train_images.reshape((60000, 28 * 28))
```

Tensör şeklini değiştirme satır ve sütunlarını ayarlayarak hedef şekle uygun hâle getirme anlamına gelir. Yeniden şekillendirilmiş tensör doğal olarak önceki tensör ile aynı eleman sayısına sahip olacaktır. Bunu en iyi bir örnekle anlayabiliriz:

```
>>> x = np.array([[0., 1.],
                  [2., 3.],
                  [4., 5.]])
>>> print(x.shape)
(3, 2)

>>> x = x.reshape((6, 1))
>>> x
array([[ 0.],
       [ 1.],
```



Şekil 2.9: Manifold verinin düzeltilmesi

2.4 Sinir Ağlarının Motoru (Gradyan Tabanlı Eniyileme)

Bir önceki kısımda gördüğünüz gibi her katman aşağıdaki gibi kendi girdisini dönüştürmektedir:

```
output = relu(dot(W, input) + b)
```

Bu ifade de W ve b tensörleri katmanın nitelikleridir. Katmanın *ağırlık* ya da *eğitilebilir parametreleri* olarak da adlandırılır (sırasıyla *kernel* ve *önyargı*[24] da denilebilir). Bu ağırlıklar ağın eğitim veri setinden öğrendiği bilgidir.

Başlangıçta bu ağırlık matrisleri küçük rastgele değerler (*rastgele başlatma*[25]) ile doldurulur. Elbette `relu(dot(W, input) + b)` ifadesinin W ve b rastgele olarak atanmış olması nedeniyle başlangıçta kullanışlı gösterimler öğrenmesi beklenemez. Hatta bu gösterimlere anlamsız bile denilebilir. Ama bu işlemden sonra geri bildirim sinyali ile bu ağırlıklar yeniden ayarlanacaklar. Bu kademeli ayarlamaya *eğitim* de denilmekte ve makine öğrenmesi tam da bu işlemdir.

Bir eğitim döngüsünde olanlar aşağıda sıralanmıştır. Bu adımları ihtiyaç kadar tekrar edeceğiz.

1. Eğitim örneklerinden x'ler ve bu örneklere karşılık gelen hedefler olan y'lerden oluşan bir yığın oluşturun.
2. Ağı x'lerden tahminler olan y_pred'leri elde etmek için *ileri yayılma*[26] tabi tutun.
3. Tahmin edilen y_pred ile hedef y'ler arasındaki farkı bularak kayıp değerini o yığın için hesaplayın.
4. Ağdaki tüm ağırlıkları o yığın için kaybı azaltacak şekilde yeniden ayarlayın.

---

[24] ÇN: İng. bias

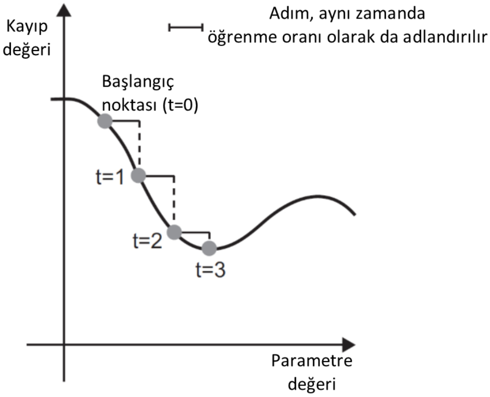
[25] ÇN: İng. random initilization

[26] ÇN: İng. feed forward

tabi tutun.

3. Tahmin edilen y_pred ile hedef y'ler arasındaki farkı bularak kayıp değerini o yığın için hesaplayın.
4. Ağın parametrelerine göre kaybın gradyanını hesapla (geriye yayılım).
5. $W- = step * gadient$ işlemini kullanarak yığın için kayıp değerini bir miktar azaltacak şekilde gradyanın tersi yönünde parametreleri güncelle.

Bu kadar basit! Aslında yukarıda tanımladığım şey *mini yığın stokastik gradyan inişiydi (mini yığın SGD)*. Stotastik terimi her veri yığını rastgele (stotastik terimi rastgelenin bilimsel eş anlamlısıdır) olarak seçildiğindendir. Şekil-2.11 ağın tek bir parametresinin olduğu ve elinizde bir eğitim örneği olduğu durumda tek boyut için nasıl olduğunu göstermektedir.

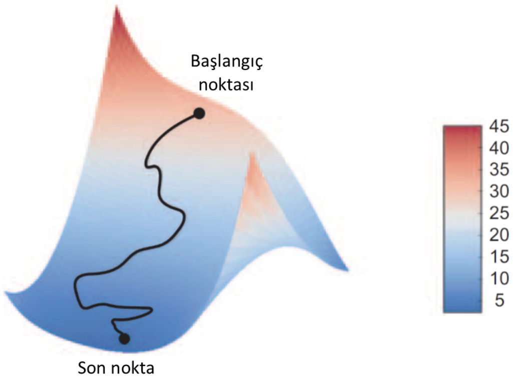


Şekil 2.11: Tek öğrenilebilir parametre için SGD

step parametresi için mantıklı bir değer seçmenin önemli olduğunu görebilirsiniz. Çok küçük bir değer seçilmesi hâlinde aşağıya doğru iniş çok fazla iterasyon gerektirecek ve lokal minimuma sıkışması da söz konusudur. Çok büyük seçildiğinde eğri üzerinde tamamen rastgele değerlere gidilebilir.

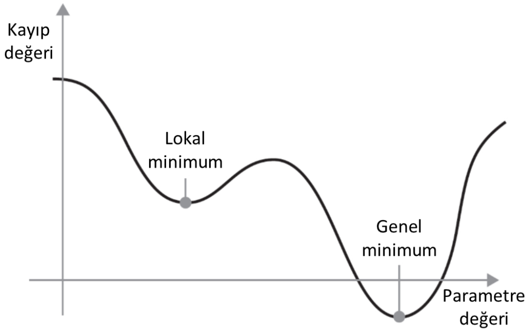
Mini yığın SGD'nin diğer bir alternatifi *gerçek* SGD olarak adlandırılan her iterasyonda tek bir veri örneği ile de çalışmak olabilir. Bir diğer alternatif ise biraz sıra dışı olsa da *yığın* SGD olarak adlandırılan eldeki tüm veri ile her adımı gerçekleştirmek olabilir. Bu işlem sonucunda her güncelleme daha gerçekçi olsa da hesaplama maliyeti çok yüksektir. Bu iki alternatif yerine daha mantıklı mini yığın büyüklüğü seçilmesi faydalı olacaktır.

Her ne kadar Şekil-2.11 tek boyutlu durumu gösteriyor olsa da gerçek uygulamalarda çok boyutlu uzaylarda gradyan inişini kullanacağız: Sinir ağlarının her bir ağırlık elemanı uzayda serbest bir noktadır ve bunlardan on binlerce hatta milyonlarca olabilir. Kayıp yüzeylerini daha iyi anlamanız için 2B uzayda gradyan iniş Şekil-2.12'de gösterilmiştir. Ama gerçekte olanın 1000000 boyutlu bir uzay olduğunu düşünürsek insanın bunu canlandırabilmesi pek de mümkün değildir. Dolayısıyla düşük boyutlu gösterimler ile kafanızda canlandırdığınızın uygulamanın da doğru sonucu vermeyebileceğini unutmayın. Bu durum derin öğrenme araştırmaları tarihi boyunca bu şekilde olmuştur.



Şekil 2.12: İki boyutlu (öğrenilen parametre sayısı iki) kayıp yüzeyinde için SGD

Buna ek olarak, SGD'nin sadece mevcut gradyan değerini göz önüne almayan aynı zamanda önceki ağırlık güncellemelerini de göz önüne alarak bir sonraki güncelleştirmeyi yapan birçok türevi vardır. Mesela Momentumlu SGD, AdaGrad, RMSProp vd. mevcuttur. Bu türevlerde kullanılan ve ilginizi hak eden *momentum* kavramı vardır. Momentum SGD kullanımında çıkan iki sorunu adreslemektedir: Yakınsama hızı ve lokal minimum. Şekil-2.13'te ağın bir parametresi için kayıp eğrisi gösterilmiştir.



Şekil 2.13: Lokal ve genel minimum

Bir parametre değerinde sola doğru gitmeye çalıştığınızda da sağa doğru gitmeye çalıştığınızda da kayıp değerinin yükseldiği *lokal minimum* bulunduğunu görebilirsiniz. Eğer SGD kullanarak olması gerekenden küçük bir parametre seçilmişse eniyileme işlemi genel minimuma ulaşamadan lokal minimuma sıkışarak (her epokta 469) yaparak kayıp enküçültülür ve ağımız kendisine verilen el yazısı rakamları yüksek bir kesinlik değeri ile sınıflandırabilecek hâle gelir. Bu noktada aslında sinir ağları hakkında bilmeniz gereken birçok konuyu öğrenmiş oldunuz.

Bölüm Özeti:

- *Öğrenme*, girdi olarak verilen eğitim örnekler ve onlara ait hedef sınıflar üzerinde kaybı enküçültecek model parametreleri kombinasyonlarını bulmaktır.
- Öğrenme, veri setinden rastgele örnekleri alarak oluşturulan yığın üzerinden yığındaki kaybı kullanarak ağ parametrelerinin gradyanları hesaplanarak sağlanır. Daha sonra a parametreleri gradyanın aksi istikametinde büyüklüğü öğrenme hızına[a] bağlı olacak şekilde bir miktar değiştirilir.
- Tüm öğrenme sürecinde, ağdaki tüm işlemlerin türevlenebilir zincirleme tensör işlemleri olması ve türevin zincir kuralının uygulanması ile mevcut yığındaki parametreler ile gradyan değerlerini eşleyen bir gradyan fonksiyonu bulunabilmektedir.
- Ağa girdileri göndermeden önce kayıp fonksiyonunun (`loss`) ve eniyileme (`optimizer`) algoritmasının belirlenmesi ilerleyen bölümlerde de sürekli karşınıza çıkacak iki konu olacaktır.
- `loss`, enküçültmek istediğiniz değer olduğundan ağın başarımını ölçmek için bir kriter olarak kullanılabilir.
- `optimizer`, kaybın gradyanının parametreleri nasıl güncelleyeceğini belirleyen yöntemdir. RMSProp, momentumlu SGD vb. olabilir.

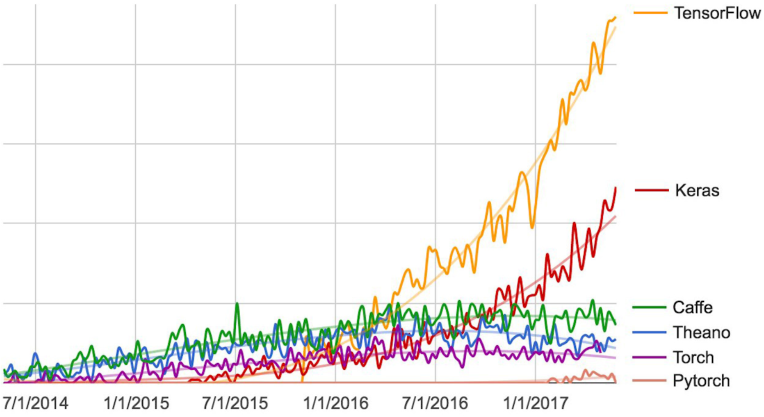
---

[a] ÇN: İng. learning rate

> Keras'ı, çekişmeli üretici ağlardan[10] sinirsel Turing makinesine kadar her türlü derin öğrenme modelinin oluşturulması için uygun hâle getirmektedir.

Keras, MIT lisansı ile dağıtılmaktadır. Yani ticari projeler dâhil her yerde serbest olarak kullanabilirsiniz. Python 2.7'den 3.6'ya kadar tüm versiyonları destekler.

Keras akademisyenlerden, küçük girişimlerdeki mühendislere, büyük şirketlerdeki mühendislerden, lisansüstü öğrencilerine ve hobi olarak uğraşanlara kadar yaklaşık 200000 kullanıcı tarafından kullanılmaktadır. Keras Google, Netflix, Uber, CERN, Yelp, Square ve yüzlerce girişimde çok farklı yelpazedeki sorunların çözümünde kullanılmaktadır. Makine öğrenmesi yarışmaları sitesi olan Kaggle platformu üzerinde de çok yaygın kullanılmaktadır ve son zamanlardaki birçok yarışmayı Keras modelleri kazanmıştır.



Şekil 3.2: Farklı derin öğrenme kütüphanelerinin Google arama motorunda aranması

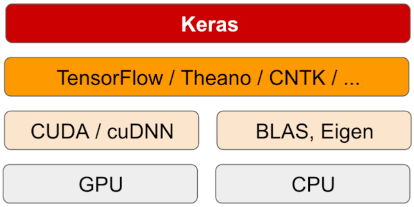
3.2.1 Keras, TensorFlow, Theano ve CNTK

Keras, kullanıcı seviyesine yakın olarak (yüksek seviyeli) derin öğrenme modelleri geliştirebilecek yapı taşlarını sunan bir kütüphanedir. Matris işlemleri, türev gibi daha düşük seviyeli işlemleri gerçekleştirmez. Bunun yerine, düşük seviyeli işlemleri yapan, iyi optimize edilmiş arka uç motoru[11] kullanır. Keras bir tensor kütüphanesi seçip Keras'ı bağlamak yerine modüler bir yol takip etmektedir (Şekil-3.3'e bakınız). Birçok farklı arka uç motoru sessizce Keras arka planında çalışmaktadır. Şu anda üç arka uç uygulaması çalışmaktadır: TensorFlow arka ucu, Theano arka ucu ve Microsoft Cognitive Toolkit (CNTK). İleride daha fazla derin öğrenme için geliştirilmiş motora destek vermesi için genişletilecektir.

---

[10] ÇN: İng. Generative adversarial networks

[11] ÇN: İng. Backend engine



Şekil 3.3: Derin öğrenmede yazılım ve donanımın beraber çalışması

TensorFlow, CNTK ve Theano derin öğrenme için günümüzde kullanılan ana platformlardır. Theano (http://www.deeplearning.net/software/theano) Montreal Üniversitesi MILA Laboratuvarı, TensorFlow (https://www.tensorflow.org/) Google ve CNTK (https://github.com/Microsoft/CNTK) Microsoft tarafından geliştirilmiştir. Keras'ta yazacağınız herhangi bir kod hiçbir değişiklik yapmadan üç kütüphane ile de çalışır: Örneğin, üçünden biri belli bir konuda daha hızlı ise isterseniz geliştirme esnasında arka uç motorunu değiştirebilirsiniz. Biz arka uç olarak daha iyi uyumlu hâle getirildiği, ölçeklenebilir olması ve ürün geliştirmeye hazır olması nedeniyle TensorFlow'un kullanılmasını tavsiye ediyoruz.

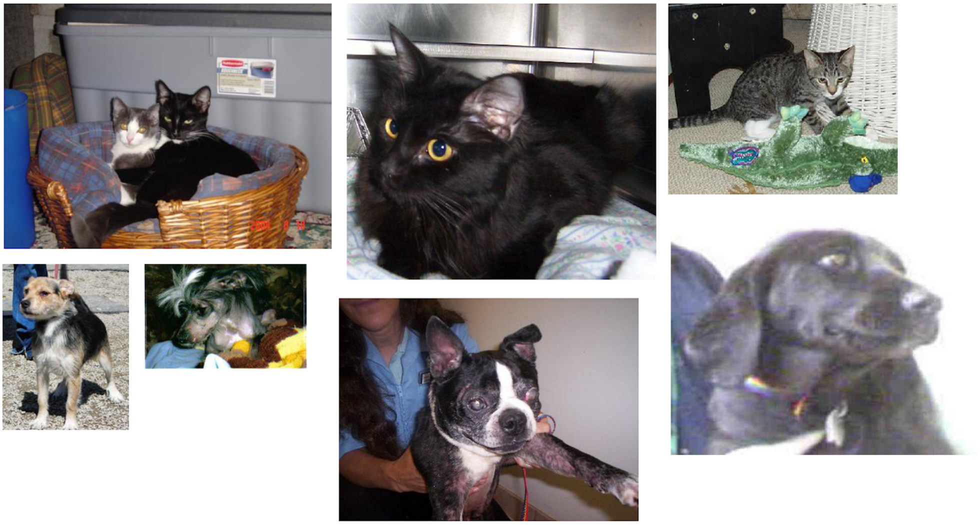
Keras, üç kütüphane ile de CPU ya da GPU üzerinde çalışabilir. TensorFlow CPU'da çalışırken tensör işlemleri için Eigen (http://eigen.tuxfamily.org) kütüphanesini kullanırken, GPU'da çalışırken NVIDIA CUDA Deep Neural Network Library (cuDNN) kullanır.

3.2.2 Keras ile Geliştirme: Hızlı Başlangıç

Hâlihazırda bir Keras modeli gördünüz: MNIST örneğinde. Keras'ta tipik iş akışı örnekte olduğu gibi şu şekildedir:

1. Eğitim verisini belirle: Girdi tensörleri ve hedef tensörleri.
2. Girdileri çıktılara eşleyen katmanlardan oluşan ağı (ya da modeli) tanımla.
3. Kayıp fonksiyonunu, eniyileme algoritmasını ve performans ölçütlerini belirleyerek eğitim sürecini tasarla.
4. `fit()` metodunu kullanarak modelinizi eğitim verisi üzerinde eğitin.

Model tanımlamanın iki yolu vardır: `Sequential` sınıfını (en sık kullanılan birbirini takip eden katmanlar şeklinde tasarımlar için) kullanmak ya da `func-`



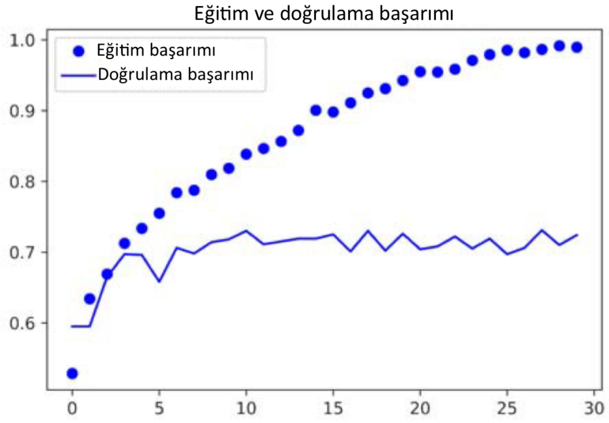
Şekil 5.8: Dogs vs. Cats veri setinden örnekler. Resim boyutları değiştirilmedi. Resimler birçok açıdan heterojendir

Bunu yapacak kod öbeği aşağıdadır.

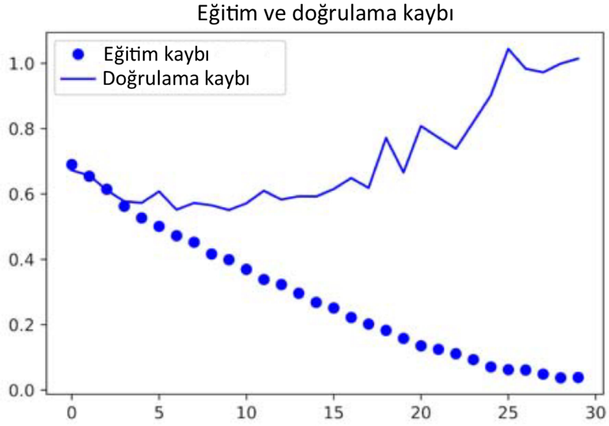
Kod 5.4 : Veri setlerini hazırlamak

```
import os, shutil

# Sıkıştırılmış dosyanın bulunduğu dizinin yolu
original_dataset_dir = '/Users/fchollet/Downloads/kaggle_original_data'


# Küçük veri setini koyacağımız dizin
base_dir = '/Users/fchollet/Downloads/cats_and_dogs_small'
os.mkdir(base_dir)

train_dir = os.path.join(base_dir, 'train')           # Eğitim setinin dizini
os.mkdir(train_dir)
validation_dir = os.path.join(base_dir, 'validation') # Doğrulama setinin  dizini
os.mkdir(validation_dir)
test_dir = os.path.join(base_dir, 'test')             # Test setinin dizini
os.mkdir(test_dir)

train_cats_dir = os.path.join(train_dir, 'cats') # Kedi eğitim setinin dizini
os.mkdir(train_cats_dir)
```

```
plt.show()
```



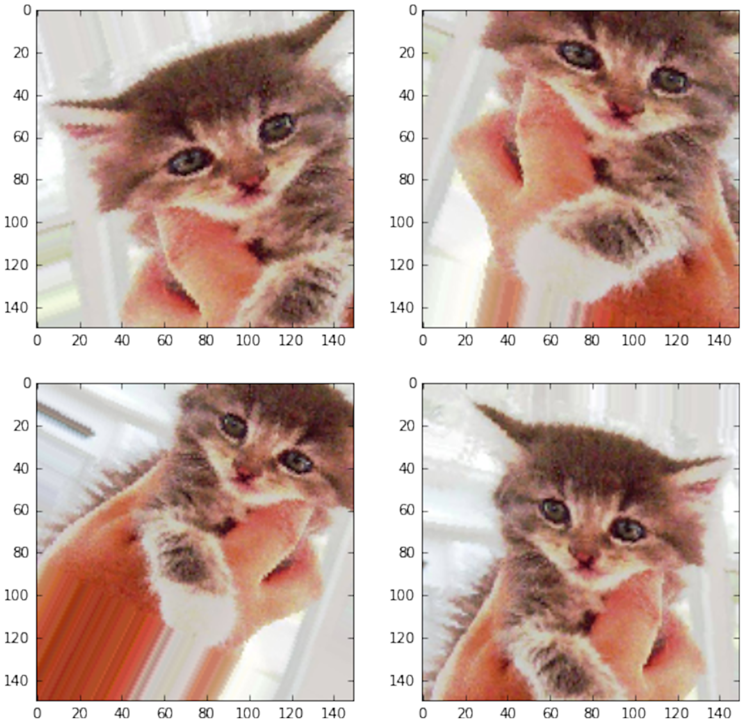
Şekil 5.9: Eğitim ve doğrulama başarımı



Şekil 5.10: Eğitim ve doğrulama kaybı

Bu şekiller aşırı uydurmanın göstergesidir. Eğitim başarımı doğrusal olarak artarak neredeyse %100'e ulaşırken doğrulama başarımı %72'de takılıp kalıyor. Doğrulama kaybı beş epokta minimum değerine ulaşıyor ve sabitleniyor. Ama eğitim kaybı doğrusal olarak düşüyor ve sıfıra yaklaşıyor.

Göreceli olarak az örnek (2000) olduğu için aşırı uydurma ilk endişelenecek hususunuz oluyor. Aslında ağırlık azalımı (L_2 düzenlileştirme) ve iletim sönümü gibi aşırı uydurma ile başa çıkabileceğiniz yolları biliyorsunuz. Ama şimdi biz bilgisayarlı görü alanında derin öğrenme uygulamalarında çok sık kullanılan yeni bir teknikten bahsedeceğiz: *Veri seti çeşitlendirme.*



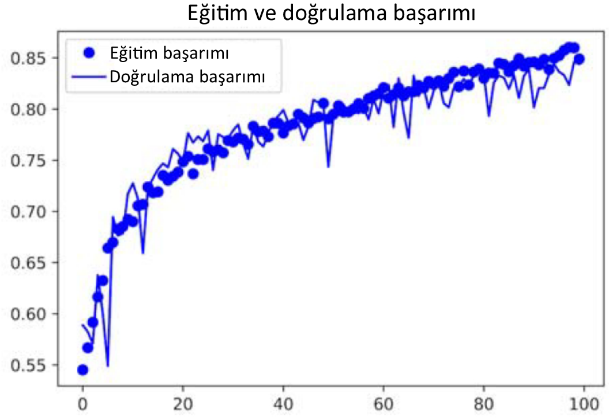
Şekil 5.11: Veri seti çeşitlendirme ile kedi resimleri oluşturmak

```
model.add(layers.Flatten())
model.add(layers.Dropout(0.5))
model.add(layers.Dense(512, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy',
              optimizer=optimizers.RMSprop(lr=1e-4),
              metrics=['acc'])
```

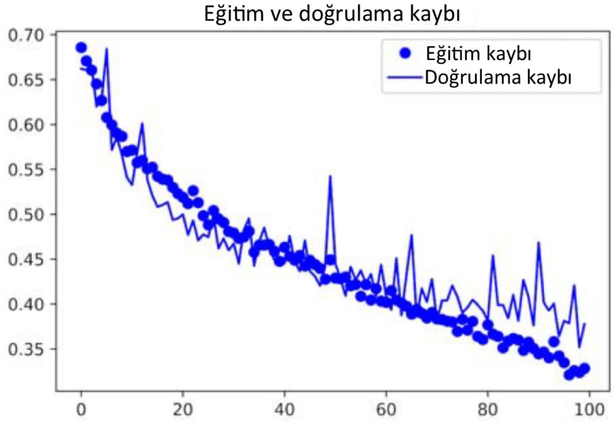
Şimdi de ağı veri çeşitlendirme ve iletim sönümü ile eğitelim:

Kod 5.14 : Evrişimli sinir ağını veri seti çeşitlendirme üreteci ile eğitmek

```
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
```



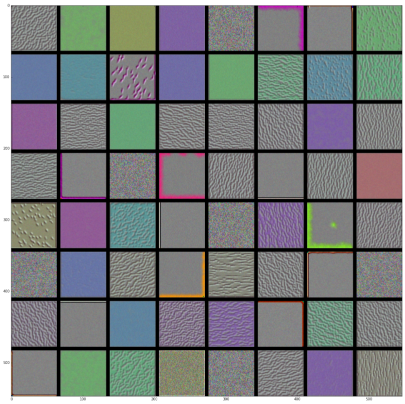
Şekil 5.12: Veri seti çeşitlendirme sonrası eğitim ve doğrulama başarımı



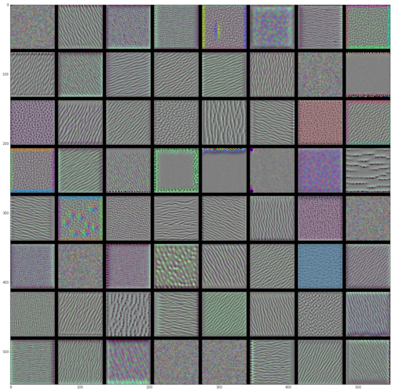
Şekil 5.13: Veri seti çeşitlendirme sonrası eğitim ve doğrulama kaybı

5.3 Öneğitimli Evrişimli Sinir Ağı Kullanmak

Resim veri seti küçük olduğunda derin öğrenmede çok yaygın kullanılan bir teknik, öneğitimli sinir ağı kullanmaktır. *Öneğitimli ağ* daha önce büyük bir veri setinde genellikle de büyük ölçekli resim sınıflandırma görevinde eğitilip kaydedilmiş ağdır. Eğer yeterince büyük ve genel bir veri setiyse uzamsal hiyerarşileri öğrenebilir ve görüntü dünyası için kullanışlı özellikler sunarak elinizdeki görev sınıflarından tamamen farklı bile olsa yeni görevlere uyarlanabilir. ImageNet veri setinde (sınıfların çoğu hayvanlar ve günlük hayatta kullandığımız nesnelerdir) eğittikten sonra çok farklı bir problemde örneğin, mobilya sınıflandırma için kullanabilirsiniz. Öğrenilen niteliklerin taşınabilir olması derin öğrenmeyi eski ve sığ öğrenme yaklaşımlarından ayıran en önemli avantajlarından biridir ve



Şekil 5.30: `block1_conv1` katmanın filtre örüntüleri



Şekil 5.31: `block2_conv1` katmanın filtre örüntüleri

Bu filtre görselleri evrişimli sinir ağlarının dünyayı nasıl gördüğünü size söylüyor: Evrişimli sinir ağının her katmanı, girdileri bir filtre kombinasyonu olarak ifade edilebilecek filtre koleksiyonu öğreniyor. Bu, Fourier dönüşümünün sinyali cosinüs fonksiyonlarına bölmesine benzer. Evrişimli sinir ağındaki filtreler giderek karmaşıklaşır ve ilerledikçe arıtılır.

- Modelin ilk katmanının `(block1_conv1)` filtreleri basit yön belirten köşeleri ve renkleri (bazı yerlerde renkli köşeler) kodlar.
- `(block1_conv1)` filtreleri köşelerin ve renklerin kombinasyonu olan dokuları kodluyor.
- İlerleyen katmanlardaki filtreler doğal resimlerde bulunan dokulara benzemeye başlıyor: Tüy, göz, yaprak vb.

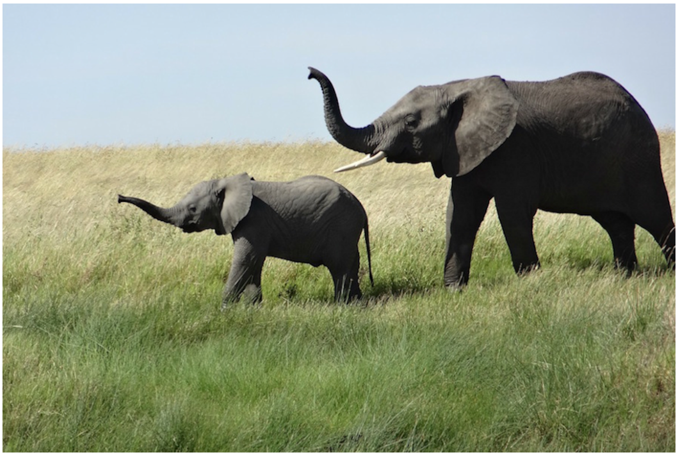
için evrişimli katmanın çıktı nitelik haritasını alır. Daha sonra bu nitelik haritasındaki her kanalın girdinin sınıfına göre gradyan değeriyle ağırlıklandırılır. Bu süreci anlamak için şöyle düşünebilirsiniz: "Sınıf tanımlanmasında herhangi bir kanalın ne kadar önemli olduğunu" bularak "girdinin farklı kanalları ne kadar aktif hâle getirdiğini" ağırlıklandırılmış uzamsal haritasından "girdinin çıktı sınıfını ne kadar yoğunlukta aktif hâle getirdiğini" gösteren uzamsal bir harita oluşturuyorsunuz.

Bu tekniği VGG16 ağını kullanarak göstereceğiz.

Kod 5.40 : Öneğitilmiş VGG16 ağını yüklemek

```
from keras.applications.vgg16 import VGG16


model = VGG16(weights='imagenet') # Daha önce sınıflandırıcıyı kullanmamıştınız.
                                  # Bu sefer kullanacaksınız.
```

Şekil-5.34'teki (Creative Commons lisanslı) iki tane Afrika filini -muhtemelen anne ve yavrusu çayırda geziniyor- düşünün. VGG16 modelinin okuyabileceği bir formata dönüştürelim: Model girdi olarak 224 × 224'lük resimler ile eğitilmiş olduğundan `keras.applications.vgg16.preprocess_input` yardımcı metot kullanarak resmi önişlemden geçirelim. Resmi okuyup 224 × 224'e yeniden boyutlandırıp `float32` Numpy tensörü hâline getirmelisiniz.



Şekil 5.34: Test resmi

Kod 5.41 : VGG16 girdisi resime önişlem yapmak

```
from keras.preprocessing import image
from keras.applications.vgg16 import preprocess_input, decode_predictions
import numpy as np
```

```
img_path = '/Users/fchollet/Downloads/creative_commons_elephant.jpg'
img = image.load_img(img_path, target_size=(224, 224))

x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
x = preprocess_input(x)
```

Şimdi öneğitimli ağı çalıştırıp tahmin vektörünü okunabilir bir formatta yazdırın:

```
>>> preds = model.predict(x)
>>> print('Predicted:', decode_predictions(preds, top=3)[0])
Predicted:, [(u'n02504458', u'African_elephant', 0.92546833),
(u'n01871265', u'tusker', 0.070257246),
(u'n02504013', u'Indian_elephant', 0.0042589349)]
```

Bu resim için en yüksek skorlu üç tahmin:

- Afrika fili (%92.54 olasılık değer ile)
- Yaban domuzu (%7.02 olasılık değer ile)
- Hint fili (%0.04 olasılık değer ile)

Ağ kaç tane olduğunu bilmeksizin bir fil olduğunu tanıdı. Tahmin vektöründeki sınıf indeksi 386 olan "Afrika fili" en büyük aktivasyonu alan sınıftır.

```
>>> np.argmax(preds[0])
386
```

Resmin hangi parçalarının "Afrika fili"ne benzetildiğini görmek için Grad-CAM süreci oluşturalım.

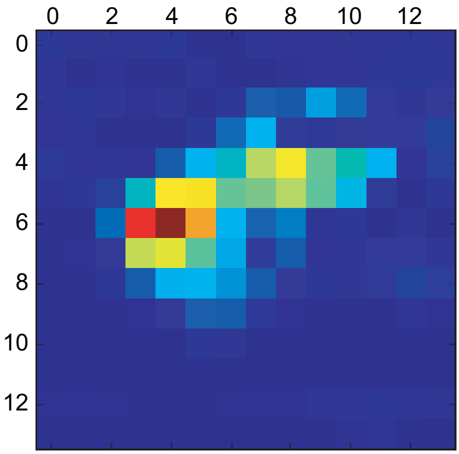
Kod 5.42 : Grad-CAM algoritmasını oluşturmak

```
# Tahmin vektöründeki "Afrika fili" girdisi
african_e66lephant_output = model.output[:, 386]

# VGG16'nın son evrişim katmanı block5_conv3'ün çıktı nitelik haritası
last_conv_layer = model.get_layer('block5_conv3')

# "Afrika fili"  sınıfının block5_conv3'ün çıktı nitelik haritasına göre gradyanı
grads = K.gradients(african_elephant_output, last_conv_layer.output)[0]

# (512,) şeklinde bir vektör. Her elemanı gradyanın nitelik haritasının
# bir kanalına göre ortalama yoğunluğu
pooled_grads = K.mean(grads, axis=(0, 1, 2))

iterate = K.function([model.input],
                     [pooled_grads, last_conv_layer.output[0]])

pooled_grads_value, conv_layer_output_value = iterate([x])

for i in range(512):
    conv_layer_output_value[:, :, i] *= pooled_grads_value[i]

heatmap = np.mean(conv_layer_output_value, axis=-1)
```

Görselleştirmek için ısı haritasını da 0 ile 1 arasına normalize edeceksiniz. Sonuçlar için Şekil-5.35'e bakabilirsiniz.

Kod 5.43 : Isı haritası son işlem

```
heatmap = np.maximum(heatmap, 0)
heatmap /= np.max(heatmap)
plt.matshow(heatmap)
```

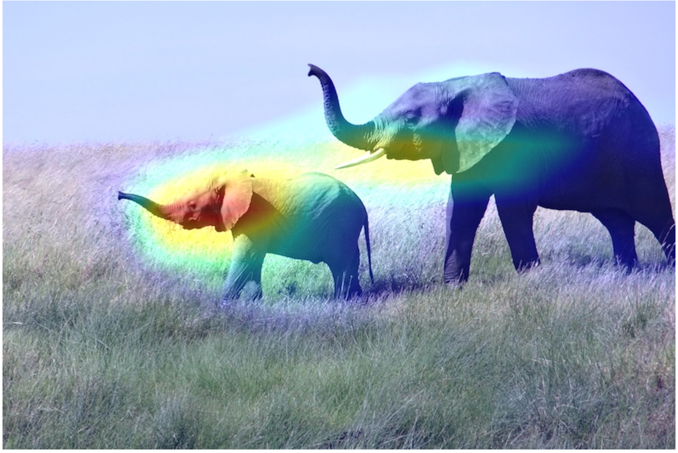


Şekil 5.35: Test resmi için Afrika fili sınıfı ısı haritası

Son olarak OpenCV kullanarak elde ettiğiniz ısı haritasını resmin üzerine uygulayın (Şekil-5.36).

Kod 5.44 : Özgün resme ısı haritasını eklemek

```
import cv2

img = cv2.imread(img_path)

heatmap = cv2.resize(heatmap, (img.shape[1], img.shape[0]))
heatmap = np.uint8(255 * heatmap)
heatmap = cv2.applyColorMap(heatmap, cv2.COLORMAP_JET)

superimposed_img = heatmap * 0.4 + img
cv2.imwrite('/Users/fchollet/Downloads/elephant_cam.jpg', superimposed_img)
```



Şekil 5.36: Sınıf aktivasyon ısı haritasının özgün resim üzerine uygulanması

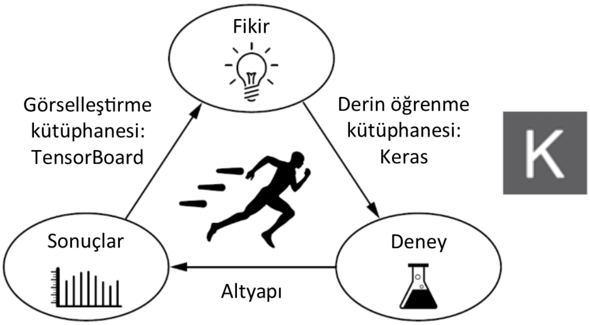
Bu görselleştirme iki önemli soruya cevap veriyor:

- Ağ neden bu resimde bir Afrika fili olduğunu düşünüyor?
- Afrika fili resimde nerede?

Dahası yavru filin kulakları oldukça aktif: Muhtemelen ağ, Hint fili ile Afrika fili arasındaki farkı bu şekilde anlıyor.

7.2.2 TensorBoard'a Giriş - TensorFlow'un Görselleştirme Kütüphanesi

İyi bir araştırma yapmak ve güzel modeller geliştirmek için deneyleriniz sırasında modelin içerisinde neler olduğu geri bildirimine ihtiyaç duyarsınız. Bu, deney yapmanın ana noktasıdır: Modelinizin ne kadar iyi performans gösterdiğine dair olabildiğince çok bilgi. İlerleme kaydetmek döngüsel ve üzerine koyarak gidilen bir süreçtir: Bir fikirle başlayıp onu deney olarak ifade edip fikrinizi doğrulamak veya yanlışlamak. Deneye başlayıp ürettiği bilgileri işleyin. Bu size bir sonraki fikir için ilham verecek. Bu süreçte her iterasyonda daha olgun ve güçlü fikirlerle geleceksiniz. Keras, fikirden deneye en kısa sürede gitmenizi, hızlı bir GPU da deneyden sonuçları olabildiğince hızlı almanızı sağlar. Ama deney sonuçlarını işlemek nedir? Burada TensorBoard devreye girmektedir.


Fikir
Görselleştirme kütüphanesi: TensorBoard
Derin öğrenme kütüphanesi: Keras
K
Sonuçlar
Deney
Altyapı

Şekil 7.9: İlerleme döngüsü

Bu bölümde, TensorFlow'la gelen tarayıcı tabanlı çalışan bir görselleştirme aracı olan TensorBoard'u tanıtıyor. Unutmayın ancak Keras'ta arka uç olarak TensorFlow kullanmanız hâlinde TensorBoard'u kullanabilirsiniz.

TensorBoard'un ana amacı eğitim esnasında modelinizde olan her şeyi gözlemlemektir. Eğer model hakkında son kaybından başka diğer bilgileri de gözlemleyebiliyorsanız, modelin ne yapıp ne yapmadığı hakkında daha doğru bir fikriniz olabilir ve böylece daha hızlı ilerlersiniz. TensorBoard tarayıcınız vasıtasıyla birçok derli toplu özelliği size sunar:

- Eğitim esnasında metrikleri görsel olarak gözlemlemek.
- Model mimarisini görselleştirmek.
- Gradyanların ve aktivasyonların histogramlarını görselleştirmek.
- 3 boyutta gömülmeleri görüp keşfedebilmek.

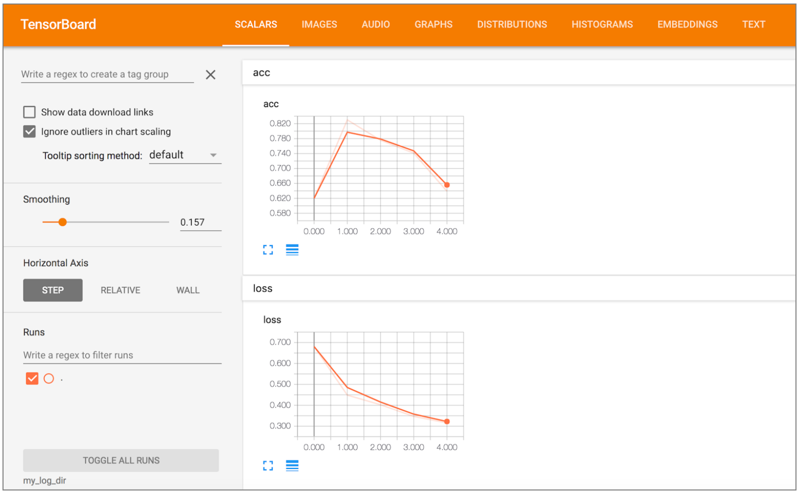
Kod 7.9 : `TensorBoard` *geri çağırmasıyla modeli eğitmek*

```
callbacks = [
    keras.callbacks.TensorBoard(
        log_dir='my_log_dir', # Kayıt kütüğü kayıt dizini
        histogram_freq=1,     # Her epoktan sonra aktivasyon histogramı kaydeder
        embeddings_freq=1,    # Gömülme verisini her epokta kaydeder
    )
]

history = model.fit(x_train, y_train,
                    epochs=20,
                    batch_size=128,
                    validation_split=0.2,
                    callbacks=callbacks)
```

Bu noktada TensorBoard sunucusunu, komut satırından kayıt kütüğü dizinini de göstererek çalıştırabilirsiniz. `tensorboard` aracı TensorFlow'u kurduğunuzda (örneğin `pip` ile) otomatik olarak kurulmuş olmalı:

```
$ tensorboard --logdir=my_log_dir
```

Şimdi tarayıcınızı açıp http://localhost:6006 adresine giderek model eğitiminize bakabilirsiniz (Şekil-7.10'a bakınız). Eğitim ve doğrulama metriklerinin canlı görüntülerine ek olarak "Histogram" sekmesinden katmanlarınızın aktivasyonlarının histogramlarına bakabilirsiniz (Şekil-7.11'e bakınız).


TensorBoard
SCALARS
IMAGES
AUDIO
GRAPHS
DISTRIBUTIONS
HISTOGRAMS
EMBEDDINGS
TEXT
Write a regex to create a tag group
Smoothing
Horizontal Axis
STEP
RELATIVE
WALL
Runs
Write a regex to filter runs
TOGGLE ALL RUNS
acc
acc
0.820
0.780
0.740
0.700
0.660
0.620
0.580
0.000
1.000
2.000
3.000
4.000
loss
loss
0.700
0.600
0.500
0.400
0.300
0.000
1.000
2.000
3.000
4.000

☐ Show data download links

☑ Ignore outliers in chart scaling

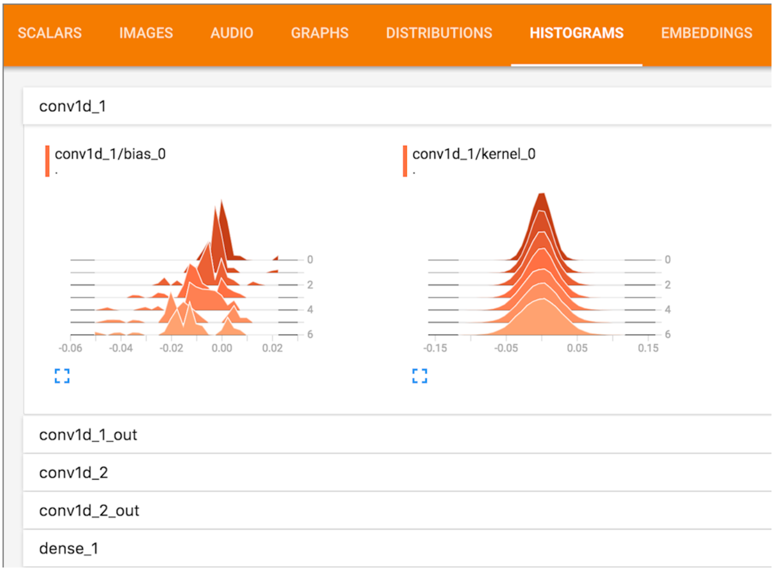
Tooltip sorting method: default

Smoothing: 0.157

☑ ○ .

my_log_dir

Şekil 7.10: TensorBoard: Metrikleri gözlemlemek

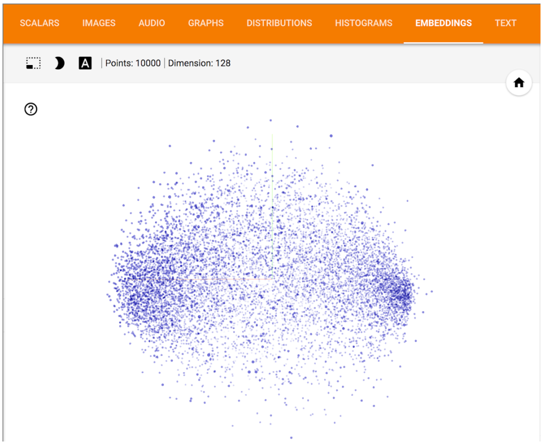

SCALARS
IMAGES
AUDIO
GRAPHS
DISTRIBUTIONS
HISTOGRAMS
EMBEDDINGS
conv1d_1
conv1d_1/bias_0
conv1d_1/kernel_0
0
2
4
6
-0.06
-0.04
-0.02
0.00
0.02
-0.15
-0.05
0.05
0.15
conv1d_1_out
conv1d_2
conv1d_2_out
dense_1

Şekil 7.11: TensorBoard: Aktivasyon histogramları

"Embeddings" sekmesi girdi sözlüğünüzdeki 10000 kelimenin `Embedding` katmanı tarafından öğrenilen gömülme yerlerini uzamsal ilişkilerini gösterir. Gömülme uzayı 128 boyutlu olduğu için TensorBoard sizin seçiminize bağlı olarak ana bileşenler analizi[14] (PSA) ya da t-distributed stochastic neighbor embedding (t-SNE) kullanarak 2 ya da 3 boyutta görmenizi sağlar. Şekil-7.12'de nokta uzayında iki ayrı küme görebilirsiniz: Pozitif çağrışımlı ve negatif çağrışımlı kelimeler. Görselleştirmeden gömülmelerin görevle beraber eğitilmesi hâlinde görev özelinde sonuçlar verdiği görülmektedir –bu yüzden genel öneğitimli kelime gömülmeleri kullanmak nadiren iyi bir fikirdir.

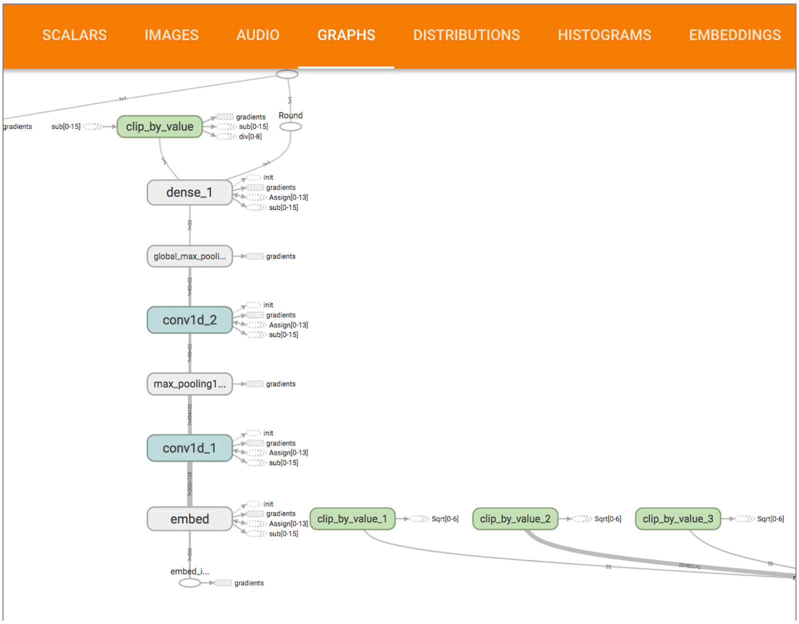
---

[14] ÇN: İng. principal component analysis


SCALARS
IMAGES
AUDIO
GRAPHS
DISTRIBUTIONS
HISTOGRAMS
EMBEDDINGS
TEXT
Points: 10000 | Dimension: 128

Şekil 7.12: TensorBoard: 3B etkileşimli kelime gömülmeleri

"Graph" sekmesi Keras modelinizin altındaki düşük seviyeli TensorFlow işlemlerinin etkileşimli bir görselleştirmesini gösterir (Şekil-7.13'e bakınız). Keras'ta modelinizi oluşturduğunuzda çok sade ve basit görünüyor olsa da sahnenin arkasında bunları gerçekleştirmek oldukça karmaşık çizgesel yapılar gerektiriyor. Gördüğünüzle değiştirdiğiniz arasındaki karışıklık farkı, ham TensorFlow'u kullanarak her şeyi sıfırdan yapmak yerine, Keras kullanmanız konusunda iyi bir motivasyon kaynağıdır. Keras tüm süreci çok basite indirgemektedir.

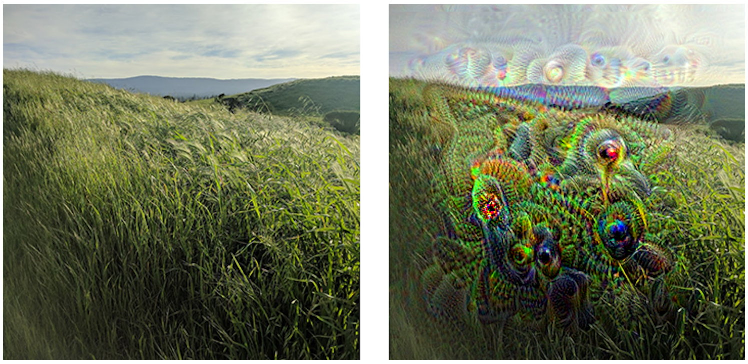

SCALARS
IMAGES
AUDIO
GRAPHS
DISTRIBUTIONS
HISTOGRAMS
EMBEDDINGS
gradients
sub[0-15]
clip_by_value
gradients
sub[0-15]
div[0-8]
Round
dense_1
init
gradients
Assign[0-13]
sub[0-15]
global_max_pooli...
gradients
conv1d_2
init
gradients
Assign[0-13]
sub[0-15]
max_pooling1...
gradients
conv1d_1
init
gradients
Assign[0-13]
sub[0-15]
embed
init
gradients
Assign[0-13]
sub[0-15]
clip_by_value_1
Sqrt[0-6]
clip_by_value_2
Sqrt[0-6]
clip_by_value_3
Sqrt[0-6]
embed_i...
gradients

Şekil 7.13: TensorBoard: Çizge görselleştirme

```
    img = np.expand_dims(img, axis=0)
    img = inception_v3.preprocess_input(img)
    return img

def deprocess_image(x): # Tensörleri tekrar geçerli bir resme dönüştüren fonksiyon
    if K.image_data_format() == 'channels_first':
        x = x.reshape((3, x.shape[2], x.shape[3]))
        x = x.transpose((1, 2, 0))
    else:
        x = x.reshape((x.shape[1], x.shape[2], 3))
    x /= 2.
    x += 0.5
    x *= 255.
    x = np.clip(x, 0, 255).astype('uint8')
    return x
```

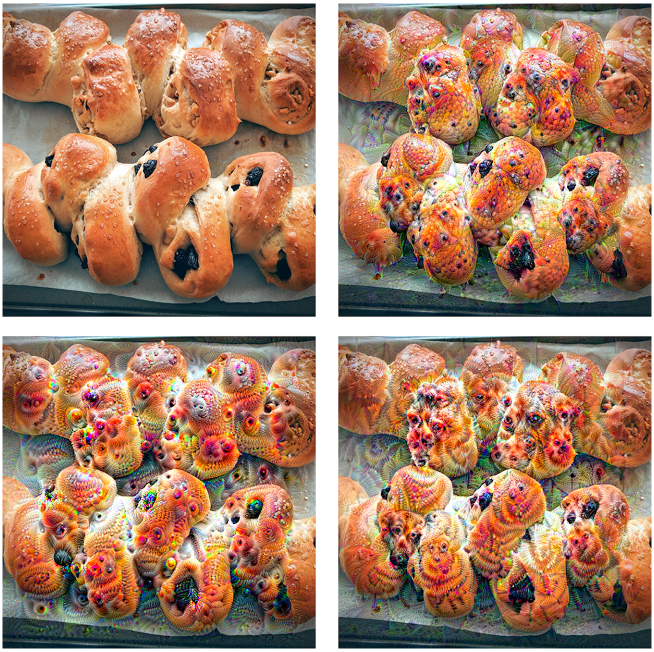
Not: Inception V3 ağnını 299 × 299 boyutunda resimlerle eğitilmiş olması nedeniyle resimler bir miktar ölçeklenmektedir. DeepDream gerçekleştirmesi genellikle 300 × 300 ile 400 × 400 boyutundaki resimlerde en iyi sonucu verse de siz istediğiniz ölçekte veya boyutta resimlerle kullanabilirsiniz.

San Francisco Koyu ile Google kampüsü arasında kalan bir bölgede çekilen küçük bir tepenin resmiyle elde edilen sonuç Şekil-8.5'te gösterilmiştir.



Şekil 8.5: DeepDream kodunu örnek bir resimde çalıştırmak

Kayba katkı yapan farklı katmanlar kullanarak elde edeceğiniz farklı sonuçları gözlemlemenizi şiddetle tavsiye ediyoruz. Ağın ilk katmanları daha bölgesel ve daha az soyut gösterimler öğrendiğinden, rüya örüntüleri daha geometrik şekillerden oluşacaktır. İlerleyen katmanların öğrendiği gösterimler ImageNet veri setinde bulunan nesnelere ait olduğundan daha çok köpek gözü, kuş tüyü gibi örüntüler olacaktır. `layer_contributions` sözlüğündeki katmanları rastgele oluşturabilirsiniz. Şekil-8.6'da ev yapımı lezzetli bir çörek kullanılarak elde edilen farklı sonuçları görebilirsiniz.



Şekil 8.6: Örnek bir resimde DeepDream için farklı yapılandırmalar denemek

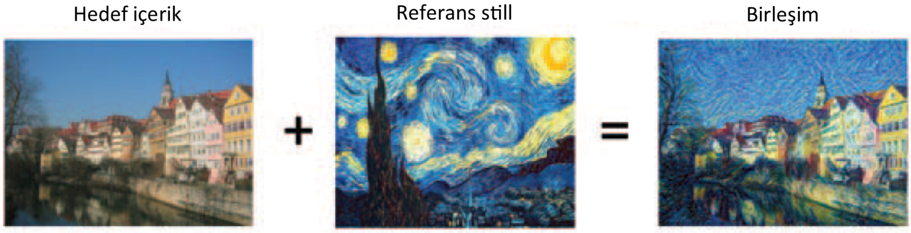
8.2.2 Özet

- DeepDream evrişimli sinir ağını tersine çalıştırıp ağın öğrendiği gösterimleri kullanarak çıktılar oluşturur.
- Sonuçlar oldukça eğlencelidir ve görsel kortekse zarar veren bazı psikolojik rahatsızlıklarda insanların gördüğünü sandığı sanrılara benzerlik göstermektedir.
- Bu süreç, sadece görsel modellere veya evrişimli sinir ağlarına özgü değildir. Konuşma, metin, müzik vb. gibi girdilere de uygulanabilir.

8.3 Sinirsel Stil Aktarımı

DeepDream'e ek olarak derin öğrenme tabanlı resmin değiştirilmesinde kullanılan bir diğer geliştirme de 2015 yazında Leon Gatys vd.[11] tarafından geliştirilen *sinirsel stil aktarımıdır*. Sinirsel stil aktarımı algoritmasının ilk çıktığı günden bu yana birçok ilerleme gerçekleştirilmiş ve birçok farklı varyasyonu geliştirilmiştir. Pek çok farklı akıllı telefon uygulamasında kullanılmıştır. Basitlik için özgün makalede anlatıldığı şekline odaklanacağız.

Sinirsel stil aktarımı, hedef resmin içeriğini koruyarak referans resmin stilini aktarmaktan ibarettir. Şekil-8.7'de bir örneğini görebilirsiniz.



Şekil 8.7: Stil aktarımı örneği

Stil aktarımı kapsamında *stil*, farklı uzamsal ölçeklerde resmin dokuları, renkleri ve görsel örüntüleri anlamına gelmektedir. *İçerik*, resimdeki makro yapılardır. Örneğin, Şekil-8.7'deki (Vincent Van Gogh'un *Yıldızlı Gece* çalışması) sarı ve mavi fırça darbeleri stil, Tübingen resmindeki binalarsa içerik olarak düşünülmelidir.

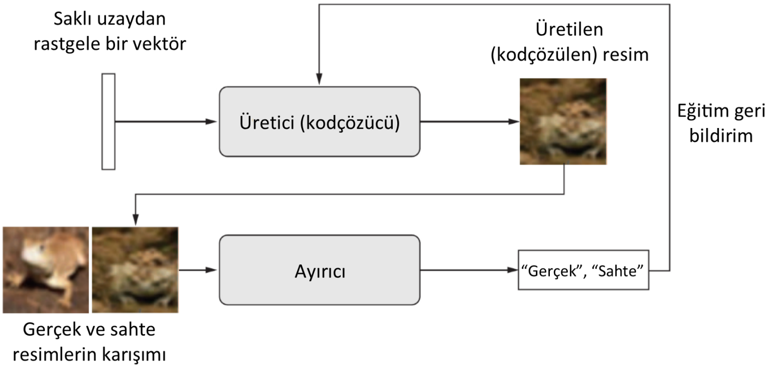
Stil aktarımı fikri doku üretilmesiyle yakından ilgilidir ve 2015 yılında sinirsel stil aktarımının ortaya çıkışından çok önce görüntü işlemede kullanılmıştır. Ama ortaya çıktığında klasik görüntü işleme tekniklerinin üretebileceklerinin yanında eşsiz sonuçlar üretebilmesiyle, görüntü işleme uygulamalarında yaratıcı bir rönesansı başlatmıştır.

Stil transferinin arkasındaki ana fikir tüm derin öğrenme algoritmalarının merkezindeki fikirle aynıdır: İstediğiniz şeyi yapması için bir kayıp fonksiyonu oluşturup onu enküçültmeye çalışırsınız. Özgün resmin içeriğini kaybetmeden referans resmin stilini uydurmak istiyorsunuz. Eğer *içerik* ve *stili* matematiksel olarak ifade edebilirsek, aşağıdaki gibi bir kayıp fonksiyonu enküçültülebilir:

```
loss = distance(style(reference_image) - style(generated_image)) +
       distance(content(original_image) - content(generated_image))
```

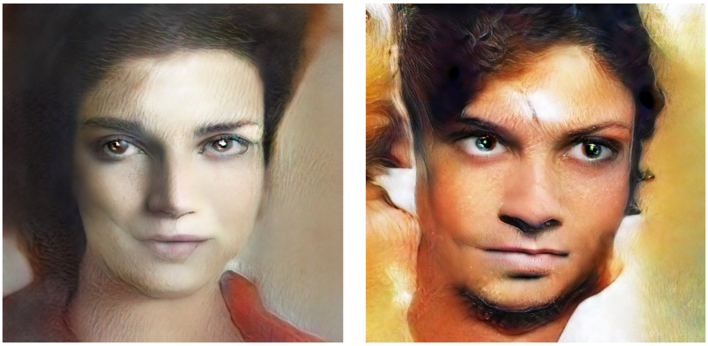
---

[11] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge, "A Neural Algorithm of Artistic Style," arXiv (2015), https://arxiv.org/abs/1508.06576.


Saklı uzaydan rastgele bir vektör
Üretilen (kodçözülen) resim
Üretici (kodçözücü)
Eğitim geri bildirim
Ayırıcı
"Gerçek", "Sahte"
Gerçek ve sahte resimlerin karışımı

Şekil 8.15: Üretici rastgele saklı vektörleri resimlere dönüştürür ve ayırıcı gerçek resimleri üretilenlerden ayırt etmeye çalışır. Üretici ayırıcıyı yanıltmak için eğitilir.

zordur ve çalışan bir GAN elde etmek, model mimarisi ve parametrelerinin çok dikkatli yapılandırılmasını gerektirir.



Şekil 8.16: Saklı uzaylılar. Resimler Mike Tyka tarafından yüz veri seti üzerinde çok aşamalı GAN kullanılarak üretilmiştir (www.miketyka.com).

8.5.1 GAN'ın Şematik Gösterimi

Bu bölümde, Keras'ta GAN'ın en basit hâliyle nasıl gerçekleneceğini anlatacağız. Çünkü GAN'lar ileri seviyeli, çok fazla teknik detaya sahip olduğundan kitabın konu kapsamının dışındadır. Derin evrişimli GAN (DCGAN) gerçeklenecektir: Üretici ve ayırıcı ağların derin evrişimli ağ olduğu bir GAN. Üreticide resimlerin boyut büyütme işlemleri `Conv2DTranspose` katmanı kullanılarak yapılacaktır.

32 × 32 boyutlarında 10 sınıfta toplam 50000 (her sınıf için 5000) örnek bulunan CIFAR10 veri setinde GAN eğiteceksiniz. İşleri basitleştirmek için

sadece "kurbağa" sınıfını kullanacaksınız.

Şematik olarak GAN'lar şöyledir:

1. `generator` ağı `(latent_dim,)` şeklindeki vektörleri `(32, 32, 3)` şeklinde resimlere eşler.
2. `discriminator` ağı `(32, 32, 3)` şeklindeki resimler için resmin gerçek olma olasılığını gösteren ikili bir skor üretir.
3. `gan` ağı üretici ve ayırıcıyı birbirine bağlar: `gan(x)= discriminator(generator(x))`.
4. Ayırıcı ağı gerçek ve sahte resimlerle ve "real"/"fake" etiketleriyle herhangi bir resim sınıflandırma modelini eğittiğiniz gibi eğiteceksiniz.
5. Üretici ağı eğitmek için üretici ağın ağırlıklarının `gan` modelinin kaybına göre gradyanlarını kullanacaksınız. Bu, her adımda üretici ağın ağırlıklarını, ürettiği resimlerin ayırıcı ağ tarafından "gerçek" kabul edeceği istikamette değiştireceğiniz anlamına gelir. Bir başka deyişle üreticiyi ayırıcıyı yanıltacak şekilde eğiteceksiniz.

8.5.2 İpuçları

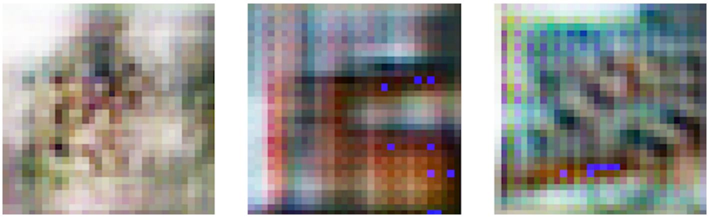
GAN'ları eğitmek ve yapılandırmak oldukça zordur. Aklınızda tutmanız gereken bazı ipuçları var. Derin öğrenmedeki birçok şey gibi bilimden ziyade, biraz simyadır: Bu ipuçları teori temelli olmaktan çok, buluşsaldır. Eldeki fenomenin sezgisel olarak anlaşılmasını sağlarlar ve her kapsamda olmasa da gözlemsel olarak iyi çalıştıkları bilinmektedir.

Bu bölümde gerçekleyeceğiniz GAN üretici ve ayırıcı ağlar için bazı ipuçları aşağıdadır. Çok ayrıntılı bir liste değil, GAN literatürünü incelediğinizde daha birçok ipucu bulabilirsiniz:

- Üretici ağın son aktivasyonunda `sigmoid` yerine modellerde yaygın olarak kullanılan `tanh` fonksiyonunu kullanacağız.
- Saklı uzaydan örnekler seçerken düzgün dağılım yerine *normal dağılım* (Gauss dağılımı) kullanacağız.
- Stokastiklik dayanıklılığı artırır. GAN eğitim sonuçları dinamik bir dengede olduğundan, GAN'lar her yönden çıkmaza girmeye müsaittir. Rastgelelik eğitim esnasında bunun olmasını engellemeye yardımcı olur. Rastgeleliği iki yoldan sağlayacağız: Ayırıcıda iletim sönümü kullanmak ve ayırıcının etiketlerine rastgele gürültü eklemek.

- Ayrık gradyanlar GAN eğitimine engel olabilir. Derin öğrenmede ayrıklık genellikle istenen bir özelliktir ama GAN'larda değil. İki şey gradyan ayrıklığına neden olabilir: En büyükleri biriktirme ve `ReLU` aktivasyonları. En büyükleri biriktirme yerine aralıklı evrişim kullanarak boyut azaltmayı ve `ReLU` yerine `LeakyReLU` katmanı kullanılmasını tavsiye ediyoruz. `ReLU`'ya benzemektedir ama küçük negatif aktivasyon değerleri ekleyerek ayrıklık kısıtlamasının çözülmesine yardım eder.

- Üretilen resimlerde satranç tahtası etkisi görülmesi, üreticinin görüntü noktası kapsamasının dengesizliğinden olur (Şekil-8.17'ye bakınız). Bunu çözmek için üretici ve ayırıcıda `Conv2DTranpose` ya da `Conv2D` kullandığımızda adım aralığıyla bölünebilir kernel boyutu kullanacağız.



Şekil 8.17: Satranç tahtası etkisi, görüntü noktalarını kapsamada dengesizlikle sonuçlanan adım aralığı ve kernel boyutunun uyumsuzluğu nedeniyle oluşur

8.5.3 Üretici

Önce, vektörleri (eğitim esnasında saklı uzaydan rastgele örneklenen) aday resimlere dönüştüren `generator` modelimizi geliştirelim. GAN'larda sıkça meydana gelen sorunlardan biri, üreticinin ürettiği resimlerin gürültüye benzemesidir. İletim sönümünü hem üretici hem de ayırıcıda kullanmak olası bir çözümdür.

Kod 8.29 : GAN üretici ağı

```python
import keras
from keras import layers
import numpy as np

latent_dim = 32
height = 32
width = 32
channels = 3
```

8.5.6 DCGAN'ı Eğitmek

Şimdi eğitime başlayabilirsiniz. Yeniden özetlersek eğitim sürecinde şematik olarak her epokta aşağıdakileri yapacaksınız:

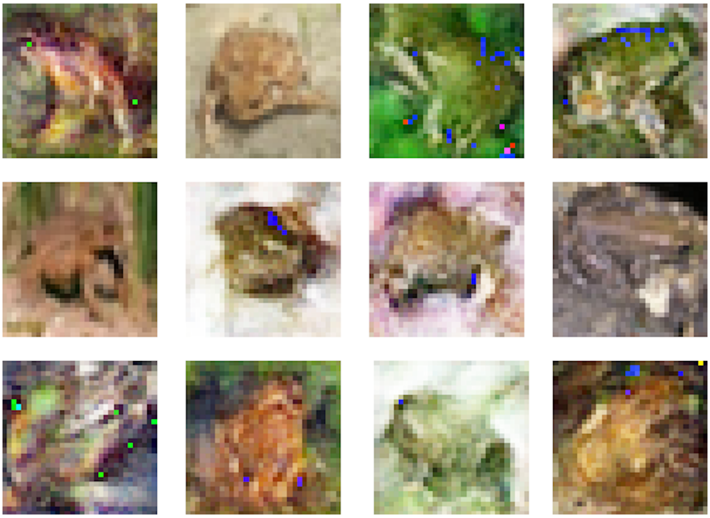
1. Saklı uzaydan rastgele noktalar çekin (rastgele gürültü).
2. Bu, rastgele gürültüyü kullanarak `generator` resimler üretecek.
3. Üretilen resimlerle gerçek resimleri birbirine karıştırın.
4. `discriminator`'ı eğitirken bu gürültü eklenmiş resimleri ve karşılık gelen "gerçek" (gerçek resimler için) ya da "sahte" (üretilen resimler için) hedefleri kullanacaksınız.
5. Saklı uzaydan yeni rastgele noktalar çekin.
6. `gan`'ı bu rastgele vektörlerle eğitin ve hedefler her zaman "bunlar gerçek resim" olsun. Ayırıcının üretilen resimler için tahminlerini "bu resimler gerçek" olacak şekilde üreticinin ağırlıklarını güncelleyin (ayırıcı donduruldu): Bu eğitim süreci, üretici kısmının ayırıcı kısmını kandırmayı öğrenmesini amaçlar.

Şimdi bunu gerçekleyelim.

Kod 8.32 : Çekişmeli üretici ağı eğitmek

```
import os
from keras.preprocessing import image

(x_train, y_train), (_, _) = keras.datasets.cifar10.load_data() #CIFAR-10 verisini yükler
x_train = x_train[y_train.flatten() == 6] #Kurbağa resimleri seçer (Sınıf numarası 6)
x_train = x_train.reshape(
    (x_train.shape[0],) +
    (height, width, channels)).astype('float32') / 255.

iterations = 10000
batch_size = 20
save_dir = 'sizin_dizininiz' # Üretilen resimlerin kayıt edileceği yer

start = 0
for step in range(iterations):
    random_latent_vectors = np.random.normal(size=(batch_size,  # Saklı uzaydan rastgele
                                                latent_dim)) # noktalar örnekler
    generated_images = generator.predict(random_latent_vectors)
```

Eğitim sırasında çekişmeli ağın kaybının çok yükseldiğini, ayırıcı ağın kaybının sıfıra yaklaştığını ve üreticiye baskın çıktığını görebilirsiniz. Böyle bir durumla karşılaştığınızda ayırıcının öğrenme oranını düşürmeyi ve iletim sönüm oranını artırmayı deneyin.



Şekil 8.18: Ayırıcıyla oyun oynamak: Her sütunda iki resim GAN tarafından üretilenlerden, bir tanesi de eğitim veri setinden geliyor. Ayırt edebilir misiniz? (Cevaplar: Sırasıyla ortadaki, yukarıdaki, alttaki ve ortadakiler gerçek.)

8.5.7 Özet

- GAN, üretici ağla ayırıcı ağın birleşiminden oluşur. Ayırıcı, üreticinin ürettiği çıktılarla eğitim veri setindeki resimleri ayırt etmek için üreticiyse, ayırıcıyı yanıltmak için eğitilir. Üretici, eğitim veri setinden hiçbir örneği doğrudan görmez, veri hakkındaki tek bilgi ayırıcıdan gelir.

- GAN'ları eğitmek zordur. Çünkü GAN eğitimi dinamik bir süreçtir ve sabit bir kayıp yüzeyinde gradyan inişi sürecinden farklıdır. GAN'ı doğru bir şekilde eğitebilmek için bazı buluşsal ipuçları vardır ve çok fazla hassas ayar gerektirir.

- GAN'lar çok gerçekçi resimler üretme potansiyeline sahiptir. VAE'lerden farklı olarak öğrendikleri saklı uzaylar sürekli ve yapısal değildir ve bu yüzden, saklı uzayın kavram vektörleriyle resim düzenleme gibi pratik uygulamalar için çok da uygun olmayabilirler.

- Metni resme eşleme.
  - *Koşullu resim üretme* –Tanıma uyan kısa bir metni resme eşlemek.
  - *Logo üretme* –Bir şirketin ismini ve tanımını logoya eşlemek.
- Resimleri resimlere eşleme.
  - *Üstün çözünürlük* –Düşük boyutlu bir resmi yüksek çözünürlüklü resme eşlemek.
  - *Görsel derinlik tahmini* –Resimde derinlik tahmini.
- Resimleri ve metni metne eşlemek.
  - *Görsel soru cevaplama* –Resimler ve resimlerin içeriğiyle ilgili doğal dil sorularına doğal dil cevapları eşlemek.
- Videoları ve metni metne eşleme.
  - *Video soru cevaplama* –Kısa videolar ve içerikleriyle ilgili doğal dil sorularına doğal dil cevapları eşlemek.

9.2 Derin Öğrenmenin Sınırları

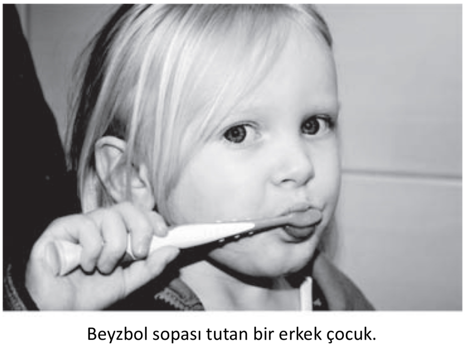
Derin öğrenme kullanarak üretilecek uygulama uzayı neredeyse sonsuzdur. Bu kadar etiketli veriye rağmen birçok uygulamaya mevcut derin öğrenmeyle ulaşmak mümkün değildir. Örneğin, bir yazılım ürününün ürün yöneticisi tarafından yazılmış yüzler, binler hatta milyonlarca İngilizce tanımı ve geliştirme takımının bu isterleri sağlayan kaynak kodları içeren bir veri seti oluşturabilirsiniz. Bu veriyle bile ürün tanımlarını okuyup onu üreten bir derin öğrenme modeli eğitemezsiniz. Bu sadece bir örnekti. Programlama ya da bilimsel bir metodu uygulamak gibi nedensellik, uzun dönem planlama ve verilerin algoritmik olarak düzenlenmesini gerektiren bir şeyi yapmak ne kadar veri olursa olsun derin öğrenmenin ulaşamayacağı noktadadır. Hatta bir sıralama algoritmasını bile derin sinir ağlarıyla öğrenmek oldukça zordur.

Bu derin öğrenme modellerinin sadece bir vektör uzayını bir diğerine eşleyen *basit zincirleme* ve *sürekli geometrik dönüşümlerden* ibaret olmasındandır. Tek yapabileceği X'ten Y'ye öğrenilebilir sürekli bir dönüşümün olduğu varsayımıyla X manifoldunu Y manifolduna eşlemektir. Derin öğrenme modelleri bir tür yazılım olarak düşünülebilecekken, tersine çoğu yazılım *derin öğrenme modeli olarak ifade edilemez* ve çoğu görevi çözebilecek bir derin öğrenme modeli olmayabilir ya da olsa bile *öğrenilebilir* olmayabilir veya yeterince veri olmayabilir.

Mevcut derin öğrenme tekniklerini kullanıp daha fazla katman ekleyerek ya da daha fazla eğitim verisi bularak bu sorunların ancak bazısını yüzeysel olarak çözebilirsiniz. Bir veri manifoldunun kesintisiz bir geometrik geçişi olarak ifade edilemeyen problemler için derin öğrenme yetersiz kalacak ve çözemeyecektir.

9.2.1 Makine Öğrenmesi Modellerini İnsanlaştırma Riski

Modern YZ ile ilgili önemli bir risk, derin öğrenme modellerinin ne yaptığının yanlış anlaşılması ve kabiliyetlerinin olduğundan çok yukarıda düşünülmesidir. İnsanın temel özelliği zihin teorisidir: Niyet okumaya meyilli olmamız, inançlarımız ve etrafımızda olanlar hakkındaki bilgimizdir. Bir kayanın üstüne gülen yüz çizmek zihnimizde onu mutlu yapar. Bunu derin öğrenmeye uyguladığımızda, resmi tanımlayan başlıklar üreten bir modeli başarıyla eğittiğimizde zihnimizde, modelin resmin içeriğini anladığını ve başlık ürettiğini düşünürüz. Sonra eğitim veri setinde başka bir resimde denediğimizde çok saçma başlıklar ürettiğini gördüğümüzde şaşırırız (Şekil-9.1'e bakınız).



Beyzbol sopası tutan bir erkek çocuk.

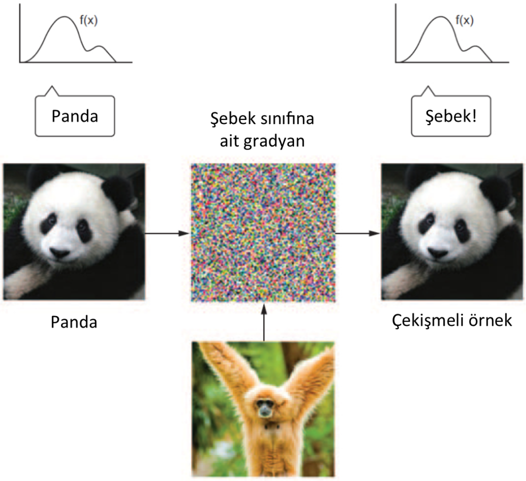
Şekil 9.1: Derin öğrenme temelli olarak resimdeki içeriği anlatan başlık görevinde başarısızlık

Bundan daha fazlası *çekişmeli örnekler*[6] tarafından karşımıza çıkarılmıştır. Çekişmeli örnekler, derin öğrenme ağlarına beslenen ve modeli, yanlış sınıflandırması için kandırmaya çalışan örneklerdir. Bunun zaten farkındasınız, örneğin, girdi uzayına gradyan çıkışı kullanarak seçilen bir evrişim filtresinin çıktıları büyütülebilir ve Bölüm-5'te bu tekniği filtrelerin görselleştirilmesinde ve Bölüm-8'de DeepDream algoritmasında gördünüz. Benzer şekilde gradyan çıkışı kullanarak resmi küçük bir şekilde değiştirerek seçilen bir sınıf için tahmini büyütebilirsiniz. Bir panda resmi alıp maymun sınıfının gradyan değerlerini eklerseniz sinir ağının pandayı maymun olarak sınıflandırmasını sağlarsınız (Şekil-9.2'ye bakınız). Bu, modellerin kırılgan ve girdi-çıktı eşleştirmelerinin insan algısından çok farklı olduğunun göstergesidir.

Kısacası, derin öğrenme modelleri girdileri hakkında bilgi sahibi değildir, en azından insan seviyesinde değildir. Biz insanların resimleri, sesleri ve dili anlaması duyu-motor tecrübelerine dayanır. Makine öğrenmesi modellerinin

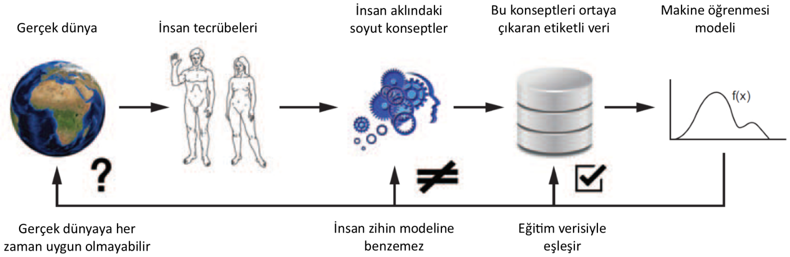
---

[6] ÇN: İng. adversarial example


f(x)
f(x)
Panda
Şebek sınıfına ait gradyan
Şebek!
Panda
Çekişmeli örnek

Şekil 9.2: Çekişmeli örnek: Resimde olabilecek gözle fark edilemeyecek kadar küçük değişimlerin modelin sınıflandırmasını baş aşağı çevirebilir.

böyle tecrübeleri yoktur ve girdilerinin insanın anladığına benzer bir yönü de yoktur. Modellerimizi insan tarafından etiketlenmiş birçok eğitim girdisiyle besleriz ve insan konseptlerini belli bir örnek kümesine eşleyen geometrik dönüşümler öğrenmelerini sağlarız ama bu eşleşme bizim zihnimizde bulunan ve tecrübeyle şekillenenin küçük bir taslağıdır. Yani aynanın karşısındaki bir nesnenin yansıması gibidir (Şekil-9.3'e bakınız).


Gerçek dünya
İnsan tecrübeleri
İnsan aklındaki soyut konseptler
Bu konseptleri ortaya çıkaran etiketli veri
Makine öğrenmesi modeli
f(x)
?
≠
☑
Gerçek dünyaya her zaman uygun olmayabilir
İnsan zihin modeline benzemez
Eğitim verisiyle eşleşir

Şekil 9.3: Şu anki makine öğrenmesi modelleri aynadaki bir yansıma gibidir

Bir makine öğrenme geliştiricisi olarak bunu aklınızda tutun ve asla sinir ağlarının yaptıkları görevi anladıklarını düşünmeyin, böyle yapmıyorlar en azından bize anlamlı gelecek şekilde yapmıyorlar. Onlar sadece öğrenmelerini istediğimiz küçük bir görevi, eğitim girdilerini eğitim hedeflerine teker teker eşleştirmeyi öğreniyorlar. Eğitim veri setinden çok farklı bir şey gösterirseniz hata yapacaklardır.

9.2.2 Sınırlı Genelleştirme ve Tamamen Genelleştirme

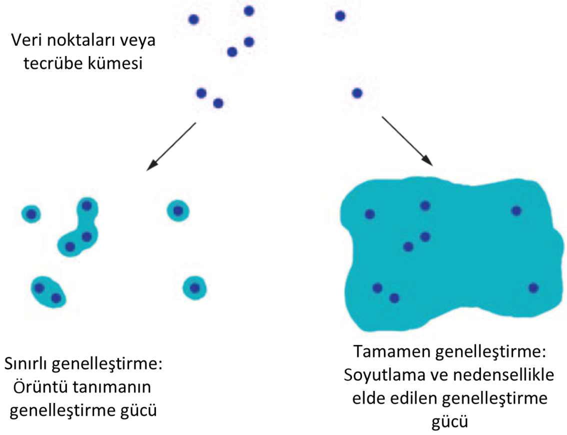
Derin öğrenme modellerinin girdileri çıktılara dönüştüren geometrik dönüşümleriyle insanların düşündüğü ve öğrendiği arasında çok temel farklılıklar vardır. Tek fark, insanların eğitim örnekleri görmek yerine kendi kendilerine tecrübeyle şekillenen bir öğrenme süreci olması değil. Farklı öğrenme süreçlerine ilave olarak altta yatan gösterimler arasında da bir fark var.

İnsanlar derin öğrenmenin ya da bir böceğin yaptığı gibi ani bir uyarana ani bir cevap vermekten daha ötesini yapabilecek kabiliyete sahiptir. İçinde bulunduğumuz ya da başkalarının içinde bulunduğu durumların karmaşık ve soyut modellerini kullanarak gelecekte farklı durumlarla baş edebilir ve uzun dönemli planlamalar yapabiliriz. Öğrendiğimiz konseptlerle daha önce hiç karşılaşmadığımız tecrübeleri birleştirebiliriz, örneğin, kot pantolon giyen bir at veya piyangodan para çıkması hâlinde neler yapacağımızı hayal edebiliriz.

Soyutlama ve *nedensellik* yaparak tecrübelerimizin ötesine zihinsel modelimizi öteye taşıyan hipotezlerle başa çıkabilme kabiliyeti, insan bilişsel yapısının karakteristiğini tanımlamaktadır. Bunu *tamamen genelleştirme* olarak adlandırıyorum: Çok az yeni veri ya da hiç yeni veri kullanmadan ilk defa karşılaşılan durumlara ayak uydurma yeteneği.

Bu derin sinir ağlarının yaptığı *sınırlı genelleştirmeden* (Şekil-9.4'e bakınız) tamamen farklıdır. Eğitim zamanında gelen veriden biraz farklı bir veri geldiğinde derin ağların girdiden çıktıya eşleştirmesi anlamsızlaşabilir. Bir roketin uzaya inmesi için gerekli parametrelerini öğrenme problemi hayal edin. Bu görev için denetimli veya pekiştirmeli öğrenme kullanarak eğitilen bir derin ağ kullanacak olsaydınız, binlerce hatta milyonlarca iniş verisiyle beslemek zorunda olurdunuz ve böylece yoğun bir girdi uzayına maruz bırakarak girdi uzayını çıktı uzayına uygun bir eşleştirmeyi öğretirdiniz. Biz insanlar, bunun aksine roket biliminin fiziksel modelini düşünerek soyutlamanın gücüyle bir ya da birkaç denemede tam bir çözüm ortaya koyabiliriz. Benzer şekilde bir insanın vücudunu kontrol eden ve şehirde arabalar tarafından ezilmeden yolunu bulabilmesini istediğiniz bir derin ağ geliştiriyor olsaydınız ağ, arabaların tehlikeli olduğunu ve kaçınma davranışları geliştirmesi gerektiğini anlamadan önce binlerce kez arabalar tarafından ezilip ölürdü. Daha sonra yeni bir şehre bırakıldığında bildiği birçok şeyi baştan öğrenmek zorunda olurdu. Diğer yandan insanlar hipotetik durumlarda soyut modellemenin gücüyle bir defa bile ölmeden güvenli davranışları öğrenebilir.

Kısacası makine algılamasında birçok ilerlemeye rağmen insan seviyesinde YZ'ya çok uzaktayız. Modellerimiz sınırlı genelleştirme yapabilir ve yeni durumlara uyum ancak yeni durumların geçmiş verilere benzemesi hâlinde mümkündür. Ancak insanın bilişselliği tamamen genelleştirebilir ve tamamen yeni durumlara çabucak uyum sağlar ve gelecek için uzun zamanlı planlama yapabilir.


Veri noktaları veya tecrübe kümesi
Sınırlı genelleştirme: Örüntü tanımanın genelleştirme gücü
Tamamen genelleştirme: Soyutlama ve nedensellikle elde edilen genelleştirme gücü

Şekil 9.4: Sınırlı genelleştirme ve tamamen genelleştirme karşılaştırması

9.2.3 Özet

Unutmamanız gereken şey, derin öğrenmenin şu ana kadar tek başarısı insan tarafından etiketlenen birçok veriyi kullanarak X uzayını Y uzayına eşlemekten ibarettir. Bu iyi yapıldığında pek çok önemli endüstride ezber bozan gelişmelere sebep olacak olsa da insan seviyesinde YZ'ya daha çok yol var.

Şu ana kadar konuştuğumuz sınırları ortadan kaldırmak için şu anda yaptığımız gibi girdi ve çıktı eşlemek yerine *nedensellik* ve *soyutlamaya* geçmeliyiz. Farklı durumları ve konseptleri soyut modelleme altyapısı bilgisayar programlarıdır. Daha önce makine öğrenme modellerini *öğrenebilen programlar* olarak nitelendirmiştik ancak şimdilik sadece tüm olası programların küçük, dar ve belli bir altkümesini öğrenebiliyoruz. Peki, modüler ve yeniden kullanılabilir şekilde herhangi bir programı öğrenebilecek miyiz? Sonraki bölümde buna giden yolu göreceğiz.

9.3 Derin Öğrenmenin Geleceği

Bu bölüm bir akademik araştırma programına katılmak isteyenlere ya da bireysel araştırma yapacaklara yeni ufuklar açmayı hedefliyor ve daha çok spekülatif. Derin ağlar hakkında bildiklerimize, sınırları ve mevcut yönelinen araştırma konularına bakarak orta vadede olacakları tahmin edebilir miyiz? Devamında söyleyeceklerim sadece şahsi fikirlerimden ibarettir. Elimde kristal bir küre yok ve söylediklerimin çoğu boşa çıkabilir. Bu tahminleri gelecekte tamamen gerçek olacağını beklediğim için değil, ilginç ve gerçekleşme ihtimalleri oldukları için paylaşıyorum.

Gelecek vadettiğini düşündüğün ana yönlere en yukarıdan bakacak olursak: